Saftlib Simplified API for Timing

Generated by Doxygen 1.9.7

Chapter 1

Saftbus: an interprocess communication library

1.1 Features

- A program `saftbusd` that can be run in the background (daemon) and provides services which can be accessed through a UNIX domain socket by Proxy objects.
- New services can be added by loading plugins at startup or during runtime of the daemon
- A code generator that facilitates developments of plugins for the daemon
- A command line tool `saftbus-ctl` to control the daemon, e.g. load/unload new plugins or remove service objects
- A deterministic memory allocator for real time applications

1.2 Architecture overview

A server process provides an inter process communication (IPC) channel (a UNIX domain socket) over which client processes can connect. At startup or at runtime, shared objects can be loaded by the server (plugins). These plugins contain C++ classes (driver classes) that are supposed to be shared resources, an should be accessible by all connected client processes. The plugins also contain automatically generated Service and Proxy classes with boilerplate code for the IPC data transfer. Instances of the Service classes are managed by the server. Client programs use instances of Proxy classes to access methods, properties, and signals provided by the driver classes.

The program `saftbusd` is a simple implementation of such a server and is installed together with the saftbus library. It is also possible to develop a different server by using saftbus library components. The program saftbus-gen can be used to generate the Proxy and Service classes from a given driver class declaration. Driver classes are ordinary C++ classes.

1.3 History

saftlib was originally written with D-Bus for inter process communication. D-Bus was later replaced with a more real-time friendly system called saftbus. Initially, the saftbus API was identical to the D-Bus API (Gio::Dbus) to avoid code changes in the rest of saftlib. Now, saftbus is re-written from scratch with a new API, and the rest of saftlib was changed to work with the new API. The re-written version of saftbus still shares some properties with D-Bus:

- A daemon running in the background maintains a list of available services.
- Services offer an interface which consists of functions, properties (setter/getter functions), and signals.
- Multiple processes can share services and use the provided interface.

Many aspects are different from D-Bus. The most important being:

- Services cannot be added by processes connecting to the daemon but have to be added by dynamically loading a plugin (shared library) into the daemon.
- Services are only usable if the signature of the interface is known. Introspection is not possible.

1.4 Environment variables

- SAFTBUS_SOCKET_PATH : determines the location of the UNIX domain socket in the file system. Default ist /var/run/saftbus/saftbus
- SAFTD_ALLOCATOR_CONFIG : set the configuration of the deterministic memory allocator. Default value is ¨16384.128 1024.1024 64.16384¨ (see below for the meaning of the numbers)

1.5 Startup

Run the saftbusd executable. In case of an error such as cannot create socket directory↩ : /var/run/saftbus, try to run with root privileges or set the SAFTBUS_SOCKET_PATH environment variable to a location where the socket can be created (e.g. /tmp/saftbus) If saftbusd was run without arguments, there will be no services apart from the saftbus::Container. Additional services can be added by loading saftbus-plugins, e.g saftbus-ctl -l libsaftd-service.so tr0:dev/wbm0. Alternatively, plugins can be loaded at startup by passing them as command line arguments to the saftbusd executable (e.g. saftbusd libsaftd-service.so tr0:dev/wbm0).

1.5.1 Deterministic memory allocator

Three versions of the saftbusd binary are distributed:

- saftbusd contains a configurable deterministic global memory allocator. It manages memory blocks of a given size and given number. The number of blocks and their size can be configured using the SAFTD_↩ ALLOCATOR_CONFIG environment variable. The default configuration is ¨16384.128 1024.1024 64.16384¨ which means
  - 16384 blocks of size 128 bytes

  - 1024 blocks of size 1024 bytes
  - 64 blocks of size 16384 bytes Allocations larger than the larges block size fall back to the default heap allocator. Number of blocks must strictly decrease. block size must strictly increase. This is intended for real time critical applications.

- `saftbusd-srta` contains a simpler non-configurable deterministic memory allocator with hard coded block sizes.
- `saftbusd-norta` uses standard heap allocations

Details on the allocator implementation are described here

1.6 Saftbus plugins

Typical use case is to run saftbusd and load a custom plugin to provide custom services, and use custom programs that communicate with the services provided by the plugin using proxy classes. See below for a simple example.

1.6.1 Services

- A saftbus service encapsulates an instance of a driver class.
- It handles the inter process communication data and translates it into function calls that are redirected to the driver instance. The result is sent back to the calling process.
- All services are managed by the saftbus::Container class, using a unique string called ¨object path¨.
- Calling `saftbus-ctl -s` list all managed services on the `saftbus::Container`.
- The source code of the service class is (typically) generated from a Driver class using the `saftbus-gen` tool, which extracts the signatures of all annotated methods and signals.
- An instance of a service class needs to be constructed with a pointer to an instance of the corresponding driver class from which it was generated.

1.6.2 Driver classes

- Are normal C++ classes, there are no restrictions on how a driver class can be written.
- Must not be defined in the global scope (`saftbus-gen` doesn't support this yet).
- Inheritance is explicitly allowed.
- Methods and signals of the driver class which are supposed to be usable from different processes (through a Proxy object) need to be annotated with the `// @saftbus-export` comment.

1.6.3 Proxy classes

- Are generated from the definition of the Driver class, just like Service classes.
- Have all methods and signals that were annotated with the `// @saftbus-export` comment in the Driver class.
- Proxy objects (instances of Proxy classes) can be used in any program to access the corresponding Service through saftbus.
- Proxy classes are constructed using the object path of the service in the `saftbus::Container`.
- Multiple Proxy instances can share the same Service instance.
- If a Service emits a signal, all Proxy instances will receive it.

1.6.4 Entry function

- Each plugin needs an export ¨C¨ function with name `create_services`.
- The function receives a pointer to a `saftbus::Container` and a vector of strings (arguments).
- It can add service objects to the container by providing an object path and a unique_ptr of the Service class.
- The Memory of the Driver class must be managed by the `create_services` function
- A destruction callback can be registered with the service object, which gets called if the service is removed from the `saftbus::Container`

1.6.5 A very simple example plugin (examples/ex00)

1.6.5.1 Driver class

The driver class has a random number generator for integers in the range [1..6]. It is called ¨Dice¨. Dice_Proxies can create a random number using the method ¨Throw¨. The result is communicated using the signal ¨Result¨. All Dice_Proxies with a connected callback function will be notified when a new number was generated.

```
 ++
#ifndef EX00_DICE_HPP
#define EX00_DICE_HPP
#include <sigc++/sigc++.h>
#include <functional>
namespace ex00 {
  class Dice {
  public:
    // @saftbus-export
    void Throw();
    // @saftbus-export
    sigc::signal<void, int> SigResult;
    // @saftbus-export
    std::function<void(int result)> FunResult;
  };
}

#endif
```

The definition of the Throw method is in Dice.cpp:

```
 ++
#include "Dice.hpp"
#include <cstdlib>
namespace ex00 {
  void Dice::Throw() {
    int number = rand()%6+1;
    SigResult.emit(number);
    FunResult(number);
  }
}
```

Four source files (Dice_Service.hpp, Dice_Service.cpp, Dice_Proxy.hpp, Dice_Proxy.cpp) will be generated with the following call to the `saftbus-gen` tool:

```
saftbus-gen Dice.hpp
```

The method `Dice::Throw` and the signals `SigResult` and `FunResult` will be identified by `saftbus-gen` and are added to the Dice_Proxy interface.

1.6.5.2 Entry point for the plugin

The file `create_services.cpp` contains the `create_services` function, which is called when the plugin is loaded by saftbusd. In this case, one argument is expected which specifies the object path of the created Dice_Service instance. Optionally, a destruction callback can be attached to the Service. This is called whenever the service is removed, (e.g. someone calling `saftbus-ctl -r <object-path>`). In this case the Driver instance is destroyed by the callback function so that the plugin can be safely removed (`saftbus-ctl -u libsaftbus-ex00-service.lo`). If the plugin is removed when there are still active services created by it, the result will be a crash of saftbusd.

```
 ++
#include "Dice.hpp"
#include "Dice_Service.hpp"
#include <saftbus/error.hpp>
#include <vector>
#include <string>

std::unique_ptr<ex00::Dice> dice;

void destroy_callback() {
  dice.reset();
}

extern "C"
void create_services(saftbus::Container *container, const std::vector<std::string> &args) {
  if (args.size() == 1 && !dice) {
    dice = std::unique_ptr<ex00::Dice>(new ex00::Dice);
    container->create_object(args[0],
      std::unique_ptr<ex00::Dice_Service>(new ex00::Dice_Service(dice.get(), destroy_callback))
    );
  } else {
    throw saftbus::Error(saftbus::Error::INVALID_ARGS, "need object path as argument");
  }
}
```

All Service classes and the create_services function are compiled into a share library (the plugin that will be loaded by saftbusd) All Proxy classes are compiled into another library that other programs can use to create Proxy objects to utilize the Service objects in saftbusd.

1.6.5.3 Utilize the Driver_Proxy

Programs instantiate a Dice_Proxy object and use it in the very same way as an instance of the Dice driver class would be used. A program which can be used for both, listen to Result as well as initiate a Throw, could look like this:

```
 ++
#include "Dice_Proxy.hpp"
#include <saftbus/client.hpp>
#include <sigc++/sigc++.h>
#include <functional>
#include <string>

std::string cmd, object_path;
void print_result_sig(int result) {
  std::cout « "dice " « object_path « " was thrown. SigResult = " « result « std::endl;
}
void print_result_fun(int result) {
  std::cout « "dice " « object_path « " was thrown. FunResult = " « result « std::endl;
}
int main(int argc, char *argv[]) {
  if (argc != 3) {
    std::cout « "usage: " « argv[0] « " <object-path> throw|listen" « std::endl;
    return 1;
  }
  object_path = argv[1];
  cmd         = argv[2];

  auto dice = ex00::Dice_Proxy::create(object_path);

  dice->SigResult.connect(sigc::ptr_fun(print_result_sig));
  dice->FunResult = &print_result_fun;
  if (cmd == "throw") {
    dice->Throw();
  }
  for (;;) {
    saftbus::SignalGroup::get_global().wait_for_signal();
    if (cmd != "listen") break;
```

```
  }
  return 0;
}
```

After launching the saftbusd with the newly created library as argument arguments of the ceate_services function must appear directly behind the shared object filename

```
$ saftbusd libex00-service.so /my/dice
```

A call to `saftbus-ctl -s` should show the newly created Dice under the `/my/dice` object path, next to the Container interface under `/saftbus`: All plugin shared object files and all connected client processes are listed as well:

```
$ saftbus-ctl -s
objects:
  object-path ID [owner] sig-fd/use-count interface-names
  /saftbus 1 [-1]  6/1 Container
  /my/dice 2 [-1]  Dice

active plugins:
  libsaftbus-ex00-service.so

connected client processes:
  5 (pid=284490)
$
```

A dice throw can be initiated by calling

```
$ ex00-ctl /my/dice throw
dice /my/dice was thrown. Result = 5
$ ex00-ctl /my/dice throw
dice /my/dice was thrown. Result = 4
$ ex00-ctl /my/dice throw
dice /my/dice was thrown. Result = 1
$
```

If other instances of ex00-ctl were launched before with the ¨/my/dice listen¨ arguments, they all would show the following:

```
$ ex00-ctl /my/dice listen
dice /my/dice was thrown. Result = 5
dice /my/dice was thrown. Result = 4
dice /my/dice was thrown. Result = 1
```

1.6.6 General rules for drivers and services

- if a driver class `Parent` is used as a saftbus service and manges other driver classes of type `Child` that are also used as saftbus services, the destructor of Parent must destroy the Child objects before removing the Child services. This is important because destructors of child objects may send signals to clients and that is only possible if the service object still exists.

1.6.7 Further information

A More complex examples which make use of inheritance and creation of nested services can be found under

- examples/ex01_dice.
- examples/ex02_dice_cup.
- the saftlib plugin saftlib

1.6.8 Allocator implementation

Each allocator manages a fixed number of fixed size memory blocks and a stack. Each stack position refers to a block and each block refers back to that stack position. Allocations return the block where the last free stack position points to (runtime is O(1)). A free operation can happen on any block, therefore pointers have to be swapped in order to remove the gaps in the stack (runtime is still O(1): always four indirections and two swap operations) The following picture explains how it works (green is free, red is allocated).

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 saftbus Namespace Reference

classes and functions of the saftbus interprocess communication library.

Classes

- class Allocator
- class ChunckAllocatorRT
- struct Client
- class ClientConnection

  Establish a connection to a running saftbus::ServerConnection using a named socket (¨/var/run/saftbus/saftbus¨ by default)
- class Container

  A Container of Service objects.
- class Container_Proxy

  A Container of Service objects.
- class Container_Service

  created by saftbus-gen from class Container and copied here
- class Deserializer

  Simple deserializer.
- class Error
- class IoSource

  An event source that is ready whenever a certain condition on a given file descriptor is met (usually POLLIN or POLLOUT)
- class LibraryLoader

  The class is constructed with the name of a shared library file (.so). It opens the file, gets the address of the function symbol ¨create_services¨. The public member function create_services forwards all arguments to the loaded function from the shared library.
- class Loop

  an event loop, driven by Sources
- class Proxy

  Base class of all Proxy objects.
- struct SaftbusInfo

  contains all information about the status of a saftbus server.
- struct SerDesAble

custom types can be sent over saftbus if they derive from this class and implement serialize and deserializ methods

- class Serializer

  Simple serializer.
- class ServerConnection

  provide a single named UNIX domain socket in the file system and handle client request on that socket
- class Service

  base class of all saftbus Services
- class SignalGroup

  Manage incoming saftbus signals and distribute them to the connected Proxy objects.
- class Source

  Base class of all event sources in a saftbus::Loop.
- class SourceHandle

  unique identifier for an event source in a saftbus::Loop
- class TimeoutSource

  An event source that is active after a given amount of time has passed.

Typedefs

- typedef void(∗ **create_services_function**) (saftbus::Container ∗container, const std::vector< std::string > &args)

Enumerations

- enum class **FunctionResult** { **RETURN** , **EXCEPTION** }

Functions

- Allocator ∗ **get_allocator** ()
- bool **operator==** (const std::unique_ptr< Source > &lhs, const SourceHandle &rhs)
- int **write_all** (int fd, const char ∗buffer, int size)
- int **read_all** (int fd, char ∗buffer, int size)
- int sendfd (int socket, int fd)

  Send given file descriptior via given socket.
- int recvfd (int socket)

  Receives file descriptor using given socket.
- bool **operator==** (const std::unique_ptr< Client > &lhs, int rhs)
- void **set_owner** ()
- void **release_owner** ()
- void **owner_only** ()
- bool **operator==** (std::pair< const unsigned int, std::unique_ptr< saftbus::Service > > &p, const int fd)

6.1.1 Detailed Description

classes and functions of the saftbus interprocess communication library.

Author

Michael Reese m.reese@gsi.de

This is a brief overview of the internal structure of the library

- Client side classes are in client.hpp and client.cpp
  - ClientConnection class: Opens a singel socket and waits for clients to connect. A connecting client first sends one part of a socketpair to the server over which all following communication happens. There is one connection per client process, that means that multiple threads within the client share one socket.
  - Proxy base class: All driver proxy classes must be derived by the Proxy base class. It provides basic services such as the protocol to register and unregister at the Server
  - Container_Proxy class: Allows remote access to the Container which stores all the Services
  - SignalGroup class: Receives signals and dispatches them to the correct Proxy object. Each Proxy object is associated with a SignalGroup. If not specified, the Proxy is associated with a default SignalGroup.
- Server side classes are in server.hpp, server.cpp and service.hpp, service.cpp
  - ServerConnection class: Manages the socket connection on the server side. It handles connection requests from ClientConnection classes. It also has a Container as member.
  - Service base class: All managed services need to derive from this class.
  - Container class: Manages all Services
  - Container_Service class: Provide remote access to the Container. Normally, Service classes are generated with the saftbus-gen tool. The Container_Service class is has a few special cases and cannot be generated by saftbus-gen.
- Data (de-)serialization happens in saftbus.hpp, saftbus.cpp
- LibraryLoader is in plugins.hpp, plugins.cpp
  - LibraryLoader class: Manages shared object files and calls the create_service entry function.
- saftbus and provides an event loop implementation and two event sources in loop.hpp, loop.cpp
  - Loop class: Implementation of an event loop.
  - Source base class: All sources used in the Loop class need to be derived from the Source base class.
  - TimeoutSource class: Execute a function once or periodically after a certain amount of time has passed.
  - IoSource class: Execute a function if a file descriptor is ready to read/write.

6.1.2 Function Documentation

6.1.2.1 recvfd()

```
int saftbus::recvfd (
            int socket )
```

Receives file descriptor using given socket.

Parameters

socket	to be used for fd recepion

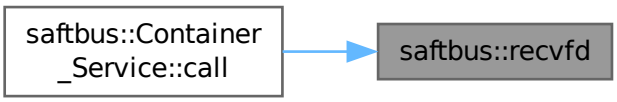
Returns

received file descriptor; -1 if failed

Note

socket should be (PF_UNIX, SOCK_DGRAM)

Here is the caller graph for this function:



6.1.2.2 sendfd()

```
int saftbus::sendfd (
            int socket,
            int fd )
```

Send given file descriptior via given socket.

Parameters

socket	to be used for fd sending
fd	to be sent

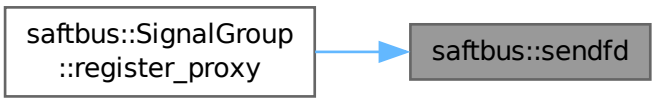
Returns

sendmsg result

Note

socket should be (PF_UNIX, SOCK_DGRAM)

Here is the caller graph for this function:



6.2 saftlib Namespace Reference

saftlib is collection of driver objects for FAIR Timing Receiver Hardware

Classes

- class ActionSink

  An output through which actions flow.
- class ActionSink_Proxy

  An output through which actions flow.
- class ActionSink_Service
- class BuildIdRom

  Representation of the SDB device with build id information. It can be used to obtain strings with gateware info and version numbers.
- class BuildIdRom_Proxy

  Representation of the SDB device with build id information. It can be used to obtain strings with gateware info and version numbers.
- class BuildIdRom_Service
- class BurstGenerator

  Burst generation service.
- class BurstGenerator_Proxy

  Burst generation service.
- class BurstGenerator_Service
- class Condition

  A rule matched against incoming events.
- class Condition_Proxy

  A rule matched against incoming events.
- class Condition_Service
- class EB_Forward

Maintains a pseudo-terminal device that mimics a serial etherbone device.

- class EB_Source

  an etherbone event source for the saftbus::Loop
- class ECA

  ECA (Event Condition Action) ECA is a hardwar unit cabable of executing actions at a given time (1 ns resolution) in response to events that meet a condition. An event contains a 64-bit timestamp, 64-bit id, 64-bit flags A condition contains a 64-bit id, 64-bit prefix-mask, 64-bit time offset Action can be one of the following:
- class ECA_Event

  ECA_Event (Event Injection interface of ECA) ECA_Event provides a method to inject events into ECA.
- class ECA_Event_Proxy

  ECA_Event (Event Injection interface of ECA) ECA_Event provides a method to inject events into ECA.
- class ECA_Event_Service
- struct ECA_OpenClose
- class ECA_Proxy

  ECA (Event Condition Action) ECA is a hardwar unit cabable of executing actions at a given time (1 ns resolution) in response to events that meet a condition. An event contains a 64-bit timestamp, 64-bit id, 64-bit flags A condition contains a 64-bit id, 64-bit prefix-mask, 64-bit time offset Action can be one of the following:
- class ECA_Service
- class ECA_TLU

  Interface to the ECA_TLU SdbInterface.
- class ECA_TLU_Proxy

  Interface to the ECA_TLU SdbInterface.
- class ECA_TLU_Service
- class EmbeddedCPUActionSink

  An output through which EmbeddedCPU actions flow.
- class EmbeddedCPUActionSink_Proxy

  An output through which EmbeddedCPU actions flow.
- class EmbeddedCPUActionSink_Service
- class EmbeddedCPUCondition

  Matched against incoming events on a EmbeddedCPUActionSink.
- class EmbeddedCPUCondition_Proxy

  Matched against incoming events on a EmbeddedCPUActionSink.
- class EmbeddedCPUCondition_Service
- class EventSource

  Interface of ECA event sources currently only implemented by saftlib::Input.
- class EventSource_Proxy

  Interface of ECA event sources currently only implemented by saftlib::Input.
- class EventSource_Service
- class FunctionGenerator

  Function Generator for creating timing triggered waveforms.
- class FunctionGenerator_Proxy

  Function Generator for creating timing triggered waveforms.
- class FunctionGenerator_Service
- class FunctionGeneratorFirmware

  Representation of the FunctionGenerator Firmware.
- class FunctionGeneratorFirmware_Proxy

  Representation of the FunctionGenerator Firmware.
- class FunctionGeneratorFirmware_Service
- class FunctionGeneratorImpl
- class Input

  Interface to a TimingReceiver input (such as button or Lemo-IO)
- class Input_Proxy

Interface to a TimingReceiver input (such as button or Lemo-IO)

- class Input_Service
- class Io

  representaion of a single IO on a TimingReceiver
- class IoControl
- class IRQ

  Represents an IRQ that is managed by saftlib.
- class LM32Cluster
- class LM32Cluster_Proxy
- class LM32Cluster_Service
- class Mailbox
- class MasterFunctionGenerator

  Interface to multiple Function Generators.
- class MasterFunctionGenerator_Proxy

  Interface to multiple Function Generators.
- class MasterFunctionGenerator_Service
- class MsiDevice
- class OpenDevice

  Holds etherbone::Device that is opened on construction and closed on destruction.
- class OpenDevice_Proxy

  Holds etherbone::Device that is opened on construction and closed on destruction.
- class OpenDevice_Service
- class Output

  An output through which on/off actions flow.
- class Output_Proxy

  An output through which on/off actions flow.
- class Output_Service
- class OutputCondition
- class OutputCondition_Proxy
- class OutputCondition_Service
- class Owned

  An object which can grant exclusive access if used in a saftbus::Container.
- class Owned_Proxy

  An object which can grant exclusive access if used in a saftbus::Container.
- class Owned_Service
- struct ParameterTuple
- class Reset
- class Reset_Proxy
- class Reset_Service
- class SAFTd

  An encapsulated etherbone::Socket with some extra features.
- class SAFTd_Proxy

  An encapsulated etherbone::Socket with some extra features.
- class SAFTd_Service
- class SCUbusActionSink

  An output through which SCUbus actions flow.
- class SCUbusActionSink_Proxy

  An output through which SCUbus actions flow.
- class SCUbusActionSink_Service
- class SCUbusCondition

  Matched against incoming events on a SCUbusActionSink.
- class SCUbusCondition_Proxy

Matched against incoming events on a SCUbusActionSink.

- class SCUbusCondition_Service
- class SdbDevice

  SdbDevices calls sdb_find_by_identity and keeps the starting address of the device registers.
- struct SearchEntry
- class SerdesClockGen
- class SoftwareActionSink

  An output through which software actions flow.
- class SoftwareActionSink_Proxy

  An output through which software actions flow.
- class SoftwareActionSink_Service
- class SoftwareCondition

  Matched against incoming events on a SoftwareActionSink.
- class SoftwareCondition_Proxy

  Matched against incoming events on a SoftwareActionSink.
- class SoftwareCondition_Service
- class TempSensor
- class TempSensor_Proxy
- class TempSensor_Service
- class Time
- class TimingReceiver

  A timing receiver.
- class TimingReceiver_Proxy

  A timing receiver.
- class TimingReceiver_Service
- class TimingReceiverAddon
- struct WalkEntry
- class Watchdog
- class WbmActionSink

  An output through which Wbm actions flow.
- class WbmActionSink_Proxy

  An output through which Wbm actions flow.
- class WbmActionSink_Service
- class WbmCondition

  Matched against incoming events on a WbmActionSink.
- class WbmCondition_Proxy

  Matched against incoming events on a WbmActionSink.
- class WbmCondition_Service
- class WhiteRabbit
- class WhiteRabbit_Proxy
- class WhiteRabbit_Service

Typedefs

- typedef saftbus::SignalGroup **SignalGroup**
- typedef boost::interprocess::allocator< ParameterTuple, boost::interprocess::managed_shared_memory↩ ::segment_manager > **ShmemAllocator**
- typedef boost::interprocess::vector< ParameterTuple, ShmemAllocator > **ParameterVector**

Functions

- int wait_for_signal (int timeout_ms=-1)
  wait for a signal from anay Proxy connected to saftbus::SignalGroup::get_global()
- void **attach_io_control** (const IoControl &io_control)
- bool **operator==** (const std::unique_ptr< ActionSink > &up, const ActionSink *p)
- void **init** (const char *leap_second_list_filename)
- int64_t **leap_second_epoch** (int n)
- int64_t **leap_second_offset** (int n)
- int64_t **UTC_offset_TAI** (uint64_t TAI)
- uint64_t **TAI_to_UTC** (uint64_t TAI)
- int **TAI_is_UTCleap** (uint64_t TAI)
- int **UTC_offset_UTC** (uint64_t UTC, int leap, int64_t *offset)
- int **UTC_to_TAI** (uint64_t UTC, int leap, uint64_t *TAI)
- int **UTC_difference** (uint64_t UTC1, int leap1, uint64_t UTC2, int leap2, int64_t *difference)
- void **test_UTC_offset** ()
- void **test_UTC_difference** ()
- void **test_conversion_forth_and_back** ()
- void **test_special_cases** ()
- Time **operator+** (const Time &lhs, const int64_t &rhs)
- Time **operator-** (const Time &lhs, const int64_t &rhs)
- Time **operator+** (const int64_t &lhs, const Time &rhs)
- Time **operator-** (const int64_t &lhs, const Time &rhs)
- Time **makeTimeUTC** (uint64_t UTC, bool isLeap)
- Time **makeTimeTAI** (uint64_t TAI)

Variables

- const char sourceVersion [ ]
- const char buildInfo [ ]
- const eb_data_t **MSI_TEST_VALUE** = 0x12345678
- const int64_t **offset_epoch_01_01_1900** = 2208988800
- const int64_t **leap_second_list** [ ][2]
- std::vector< std::array< int64_t, 2 > > **leap_second_vector**
- const int64_t **sec** = INT64_C(1000000000)
- const int64_t **msec** = INT64_C(1000000)
- const int64_t **usec** = INT64_C(1000)
- const int64_t **nsec** = INT64_C(1)

6.2.1 Detailed Description

saftlib is collection of driver objects for FAIR Timing Receiver Hardware

The classes hide direct register access and provide high level access to the hardware functionality. The code can be used directly by linking the driver library, or by loading the driver library as a plugin into a running saftbusd access them with the saftbus inter process communication library through the driver proxy classes.

In general, SDB devices on the hardware are represented by a class derived from SdbDevice class. All hardware devices, that are MSI masters are represented by a class derived from MsiDevice class. An instance of an MsiDevice class can be used to register MSI callback functions at the SAFTd instance.

6.2.2 Function Documentation

6.2.2.1 wait_for_signal()

```
int saftlib::wait_for_signal (
            int timeout_ms = -1 )
```

wait for a signal from anay Proxy connected to saftbus::SignalGroup::get_global()

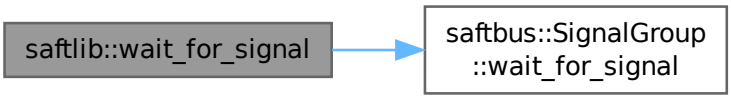
Parameters

timeout_ms	if no signal arrives return after so many milliseconds, default is -1 which means: no timeout

Returns

>0 if a signal was received, 0 if timeout was hit, < 0 in case of failure

Here is the call graph for this function:



6.2.3 Variable Documentation

6.2.3.1 buildInfo

```
const char saftlib::buildInfo
```

Initial value:

```
=
  "built by " USERNAME " on " __DATE__ " " __TIME__ " with " HOSTNAME " running " OPERATING_SYSTEM
```

6.2.3.2 sourceVersion

```
const char saftlib::sourceVersion
```

Initial value:

```
=
  PACKAGE_STRING " (" GIT_ID "): " SOURCE_DATE
```

6.3 software_tr Namespace Reference

Software emulation of a TimingReceiver.

Classes

- class BuildIdRom
- class Device
- class EBslave
- class EcaEventsIn
- class EcaQueue
- class EcaUnitControl
- class FpgaReset
- class IoControl
- class LM32ClusterInfoRom
- class MsiMailbox
- class SDBrecords
- struct SoftwareECA
- class WatchdogMutex
- class WrPpsGenerator

Enumerations

- enum **std_logic_t** {
  STD_LOGIC_U , **STD_LOGIC_X** , **STD_LOGIC_0** , **STD_LOGIC_1** ,
  STD_LOGIC_Z , **STD_LOGIC_W** , **STD_LOGIC_L** , **STD_LOGIC_H** ,
  STD_LOGIC_DASH }

Variables

- EBslave ∗ **eb_slave** = nullptr
- uint32_t **eca_msi_target_adr** = 0
- uint32_t **eca_in_buffer** [8] = {0,}
- uint32_t **eca_tag** = 0
- struct software_tr::SoftwareECA **software_eca**
- std::map< uint32_t, uint32_t > **sdb_memory**

6.3.1 Detailed Description

Software emulation of a TimingReceiver.

The purpose of this program is to run saftlib software test programs without attaching actual hardware devices to the system.

The first part of this program consists of an etherbone slave implementation that was originally developed to connect GHDL simulations to real host programs. It behaves like a USB serial device and is implemented as a pseudo-terminal in /dev/pts/<n>, where <n> is an integer that is determined by the OS. After startup, the full device name is written into the temporary file ¨/tmp/simbridge-eb-device¨, such that eb-tools can be used like this: ¨eb-ls $(cat /tmp/simbridge-eb-device)¨

The second part of this program consists of an emulation of some hardware registers in a TimingReceiver and partially of its behavior. Register layout is based on a SDB record file from a real TimingReceiver. This Program can also be used to extract such an SDB record file from existing hardware (use the –extract-sdb <device name> option and redirect the output into a file). Currently, the program implements enough TimingReceiver functionality that saftlib can connect to it (using the pseudo-terminal device). Saftlib clients can create software action sinks and inject events into the simulated hardware. The timing events are redistributed to the correct SoftwarAction↩ Sinks just as they would be if saftlib would work on real hardware. The functionality is limited in that flags such as LATE,CONFLICT,DELAYED,EARLY are not supported. Late events are always delivered immediately. The execution time comes from the system time and is not synchronized to a WhiteRabbit network.

Besides parts of the ECA, not much of the hardware behavior is currently implemented.

Chapter 7

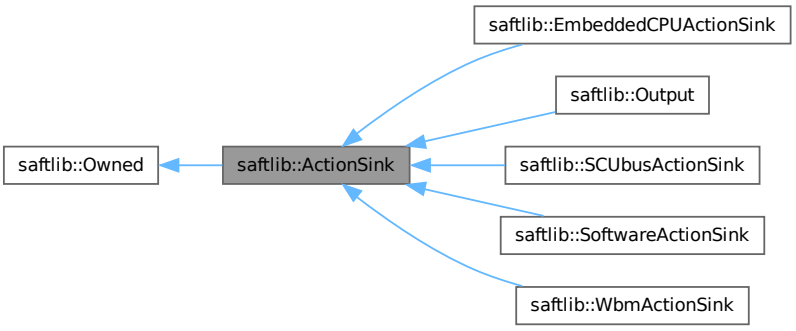
Class Documentation

7.1 saftlib::ActionSink Class Reference

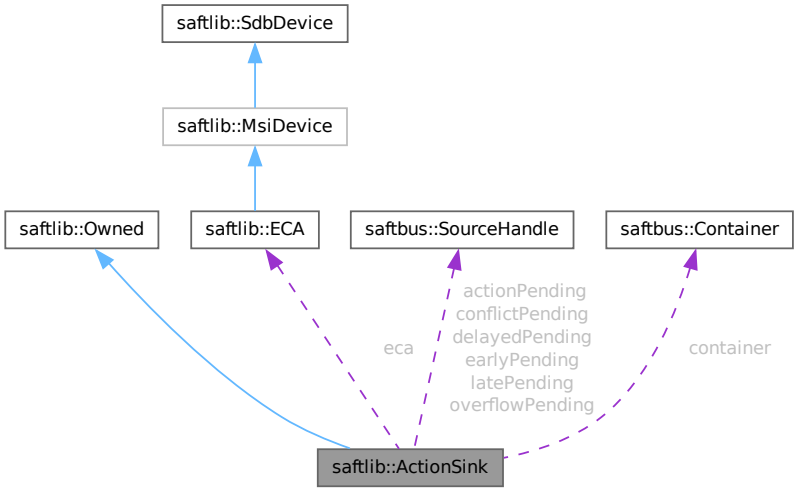
An output through which actions flow.

`#include <ActionSink.hpp>`

Inheritance diagram for saftlib::ActionSink:



Collaboration diagram for saftlib::ActionSink:



Classes

- struct Record

Public Types

- typedef std::map< unsigned, std::unique_ptr< Condition > > **Conditions**

Public Member Functions

- ActionSink (ECA &eca, const std::string &action_sink_object_path, const std::string &name, unsigned channel, unsigned num, saftbus::Container *container=nullptr)

  ActionSink constructor.
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions () const

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions () const

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions () const

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset () const

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset () const

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)

- uint64_t getLatency () const

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold () const

  Actions further into the future than this are early.
- uint16_t getCapacity () const

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull () const

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate () const

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount () const

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount () const

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount () const

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount () const

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount () const

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount () const

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)
- void **compile** ()
- const std::string & **getObjectName** () const
- const std::string & **getObjectPath** () const
- const Conditions & **getConditions** () const
- unsigned **getChannel** () const
- unsigned **getNum** () const
- virtual void **receiveMSI** (uint8_t code)
- Condition * **getCondition** (const std::string object_path)
- void **removeCondition** (Condition *condition)
- unsigned **createConditionNumber** ()
- template<typename ConditionType , typename... Args>
  std::string **NewConditionHelper** (bool active, Args &&... args)

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)

- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Attributes

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions

- Record **fetchError** (uint8_t code) const
- bool **updateOverflow** () const
- bool **updateAction** () const
- bool **updateLate** () const
- bool **updateEarly** () const
- bool **updateConflict** () const
- bool **updateDelayed** () const

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes

- std::string **object_path**
- ECA & **eca**
- std::string **name**
- unsigned **channel**
- unsigned **num**
- int64_t **minOffset**
- int64_t **maxOffset**
- std::chrono::nanoseconds **signalRate**
- uint64_t **overflowCount**
- uint64_t **actionCount**
- uint64_t **lateCount**
- uint64_t **earlyCount**
- uint64_t **conflictCount**
- uint64_t **delayedCount**
- std::chrono::steady_clock::time_point **overflowUpdate**
- std::chrono::steady_clock::time_point **actionUpdate**
- std::chrono::steady_clock::time_point **lateUpdate**
- std::chrono::steady_clock::time_point **earlyUpdate**
- std::chrono::steady_clock::time_point **conflictUpdate**
- std::chrono::steady_clock::time_point **delayedUpdate**
- uint64_t **latency**
- uint64_t **earlyThreshold**
- uint16_t **capacity**
- saftbus::SourceHandle **overflowPending**
- saftbus::SourceHandle **actionPending**
- saftbus::SourceHandle **latePending**
- saftbus::SourceHandle **earlyPending**
- saftbus::SourceHandle **conflictPending**
- saftbus::SourceHandle **delayedPending**
- Conditions **conditions**
- saftbus::Container * **container**

7.1.1 Detailed Description

An output through which actions flow.

Conditions created on this ActionSink specify which timing events are translated into actions. These actions have execution timestamps which determine when the action is to be executed, precise to the nanosecond.

More specialized versions of this interface provide the 'NewCondition' method to create conditions specific to the type of ActionSink. For example, SoftwareActionSinks create conditions that emit signals to software. This interface captures the functionality common to all ActionSinks, such as atomic toggle, offset constraints, and introspection. In particular, ActionSinks have common failure modes.

Actions are queued for delivery at the appropriate time, in hardware. Hardware has limited storage, reflected by the Fill, Capacity, and MostFull properties. These should be monitored to ensure that the queue never overflows.

The first failure mode of an ActionSink is that the queue overflows. In this case, the hardware drops an action. Obviously, this is a critical error which may result in an undefined state. To prevent these failures, MostFull should be kept below some safety margin with respect to Capacity. Note: several distinct ActionSinks may share underlying hardware, and the Fill property is shared amongst all instances. Each overflow is recorded in the OverflowCount register. Due the rate at which this counter might increase, the API throttles updates using the SignalRate property.

Another critical error for an ActionSink is the possibility of a late action. This indicates that hardware was instructed to execute an action at a time in the past. This is typically caused by either a malfunctioning data master, desynchronized clocks, or conditions with large negative offsets. This is a critical failure as it might leave the system in an undefined state. Conditions may be configured to either drop or execute late actions. If late actions are dropped, a magnet might never be turned off. If late actions are executed, a magnet might be turned on again after it was supposed to be turned off (ie: the actions get misordered). In any case, LateCount is increased.

Similar to late actions, one can also have early actions. If an action is scheduled for execution too far into the future, the timing receiver will choose to mark it as early. This prevents these actions from permanently consuming space in the finite hardware buffers. Early actions are also critical failures as it can leave the system in an undefined state, just as a late action. Conditions may be configured to either drop or execute early actions.

The final misordering failure for an ActionSink is the possibilty of a Conflict. If two actions are scheduled to be executed at the same nanosecond, their relative order is undefined. These conflicts are a critical error as they may leave the system in an undefined state, just as with early and late actions. Conflicts should be prevented by never creating two Conditions on the same ActionSink which could occur at the same time. Note that it is NOT a Conflict for two actions to be executed at the same time by two different ActionSinks. For software, this means that two programs, each with their own SoftwareActionSink do not need to be concerned about conflicts between their schedules. As another example, two different LEMO output cables (corresponding to two OutputActionSinks) can be toggled high at the same time.

Finally, these is the possibility of a delayed action. Unlike late, early, and conflicting actions, delayed actions are never misordered. Thus, delayed actions are typically not as severe a failure mode, and Conditions default to allowing their execution. Delays can happen when the delivery rate of actions (potentially 1GHz) exceeds the capability of the receiver to process the actions. For example, an output might require 100ns to deliver an action. If two actions are scheduled for delivery back-to-back with 1ns between, the second action is delayed. For Software↩ ActionSinks, delays can probably always be ignored because the handler is much much slower than the hardware. For a kicker, on the other hand, a delay is probably a critical error.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ActionSink()

```
saftlib::ActionSink::ActionSink (
              ECA & eca,
              const std::string & action_sink_object_path,
              const std::string & name,
              unsigned channel,
              unsigned num,
              saftbus::Container * container = nullptr )
```

ActionSink constructor.

Parameters

eca	points to the ECA object on which the ActionSink is created
name	name of the ActionSink
channel	the ECA channel that feeds the ActionSink
num	subchannel of the ECA channel

Create a new action sink on an ECA device with a given name. If the name is an empty string, a name in the form ¨_<number>¨ will be generated. The ActionSink is attached to the given ECA channel/subchannel.

7.1.3 Member Function Documentation

7.1.3.1 getActionCount()

```
uint64_t saftlib::ActionSink::getActionCount ( ) const
```

The number of actions processed by the Sink.

Returns

The number of actions processed by the Sink.

As actions can be emitted very rapidly, ActionCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.1.3.2 getActiveConditions()

```
std::vector< std::string > saftlib::ActionSink::getActiveConditions ( ) const
```

All active conditions created on this ActionSink.

Returns

All active conditions created on this ActionSink.

All active conditions on the ActionSink. Each object path implements the the matching Condition interface; for example, a SoftwareActionSink will have SoftwareConditions.

7.1.3.3 getAllConditions()

```
std::vector< std::string > saftlib::ActionSink::getAllConditions ( ) const
```

All conditions created on this ActionSink.

Returns

All conditions created on this ActionSink.

All active and inactive conditions on the ActionSink. Each object path implements the the matching Condition interface; for example, a SoftwareActionSink will have SoftwareConditions.

7.1.3.4 getCapacity()

```
uint16_t saftlib::ActionSink::getCapacity ( ) const
```

The maximum number of actions queueable without Overflow.

Returns

The maximum number of actions queueable without Overflow.

The timing receiver hardware can only queue a limited number of actions. This property reports the maximum number of actions that may be simultaneously queued. Be aware that this resource may be shared between multiple ActionSinks. For example, all SoftwareActionSinks share a common underlying queue in hardware. This Capacity represents the total size, which must be shared.

7.1.3.5 getConflictCount()

```
uint64_t saftlib::ActionSink::getConflictCount ( ) const
```

The number of actions which conflicted.

Returns

The number of actions which conflicted.

If two actions should have been executed simultaneously by the same ActionSink, they are executed in an undefined order. Each time this happens, the ConflictCount is increased. As conflicts can occur very rapidly, ConflictCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.1.3.6 getDelayedCount()

```
uint64_t saftlib::ActionSink::getDelayedCount ( ) const
```

The number of actions which have been delayed.

Returns

The number of actions which have been delayed.

The timing receiver emits actions potentially every nanosecond. In the case that the receiver cannot immediately process an action, the timing receiver delays the action until the receiver is ready. This can happen either because the receiver was still busy with a previous action or the output was used externally (bus arbitration). As actions can be emitted very rapidly, DelayedCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.1.3.7 getEarlyCount()

```
uint64_t saftlib::ActionSink::getEarlyCount ( ) const
```

The number of actions delivered early.

Returns

The number of actions delivered early.

As described in the interface overview, an action can be early due to a buggy data master, loss of clock synchronization, or very positive condition offsets. This is a critical failure as it can result in misordering of executed actions. Each such failure increases this counter.

As early acitons can occur very rapidly, EarlyCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.1.3.8 getEarlyThreshold()

```
uint64_t saftlib::ActionSink::getEarlyThreshold ( ) const
```

Actions further into the future than this are early.

Returns

Actions further into the future than this are early.

If an action is scheduled for execution too far into the future, it gets truncated to at most EarlyThreshold nanoseconds into the future.

7.1.3.9 getInactiveConditions()

```
std::vector< std::string > saftlib::ActionSink::getInactiveConditions ( ) const
```

All inactive conditions created on this ActionSink.

Returns

All inactive conditions created on this ActionSink.

All inactive conditions on the ActionSink. Each object path implements the the matching Condition interface; for example, a SoftwareActionSink will have SoftwareConditions.

7.1.3.10 getLateCount()

```
uint64_t saftlib::ActionSink::getLateCount ( ) const
```

The number of actions delivered late.

Returns

The number of actions delivered late.

As described in the interface overview, an action can be late due to a buggy data master, loss of clock synchronization, or very negative condition offsets. This is a critical failure as it can result in misordering of executed actions. Each such failure increases this counter.

As late actions can occur very rapidly, LateCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.1.3.11 getLatency()

```
uint64_t saftlib::ActionSink::getLatency ( ) const
```

Nanoseconds between event and earliest execution of an action.

Returns

Nanoseconds between event and earliest execution of an action.

7.1.3.12 getMaxOffset()

```
int64_t saftlib::ActionSink::getMaxOffset ( ) const
```

Maximum allowed offset (nanoseconds) usable in NewCondition.

Returns

Maximum allowed offset (nanoseconds) usable in NewCondition.

Large offsets are almost always an error. A large positive offset could result in early actions being created. By default, no condition may have an offset larger than 1s. Attempts to create conditions with offsets greater than MaxOffset result in an error. Change this property to override this safety feature.

7.1.3.13 getMinOffset()

```
int64_t saftlib::ActionSink::getMinOffset ( ) const
```

Minimum allowed offset (nanoseconds) usable in NewCondition.

Returns

Minimum allowed offset (nanoseconds) usable in NewCondition.

Large offsets are almost always an error. A very negative offset could result in Late actions. By default, no condition may be created with an offset smaller than -100us. Attempts to create conditions with offsets less than MinOffset result in an error. Change this property to override this safety feature.

7.1.3.14 getMostFull()

```
uint16_t saftlib::ActionSink::getMostFull ( ) const
```

Report the largest number of pending actions seen.

Returns

Report the largest number of pending actions seen.

The timing receiver hardware can only queue a limited number of actions. This property reports the highest Fill level seen by the hardware since it was last reset to 0. Keep in mind that the queue may be shared, including actions from all users of the underlying hardware. So two programs each using a SoftwareActionSink will potentially see an increase in this value when both programs are active.

7.1.3.15 getOverflowCount()

```
uint64_t saftlib::ActionSink::getOverflowCount ( ) const
```

The number of actions lost due to Overflow.

The underlying hardware queue may overflow once Fill=Capacity. This is a critical error condition that must be handlded. The causes may be either: 1- the actions have an execution time far enough in the future that too many actions are buffered before they are executed, or 2- the receiving component is unable to execute actions as quickly as the timing system delivers them. The second case is particularly likely for SoftwareActionSinks that attempt to listen to high frequency events. Even though SoftwareActionSinks share a common queue, Overflow is reported only to the ActionSink whose action was dropped. As overflows can occur very rapidly, OverflowCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.1.3.16 getSignalRate()

```
uint64_t saftlib::ActionSink::getSignalRate ( ) const
```

Minimum interval between updates (nanoseconds, default 100ms).

Returns

Minimum interval between updates (nanoseconds, default 100ms).

The properties OverflowCount, ActionCount, LateCount, EarlyCount, ConflictCount, and DelayedCount can increase rapidly. To prevent excessive CPU load, SignalRate imposes a minimum cooldown between updates to these values.

7.1.3.17 ReadFill()

```
uint16_t saftlib::ActionSink::ReadFill ( )
```

Report the number of currently pending actions.

Returns

Number of pending actions.

The timing receiver hardware can only queue a limited number of actions. This method reports the current number of queued actions, which includes actions from all users of the underlying hardware. So two programs each using a SoftwareActionSink will potentially see an increase in this value when both programs are active. Also, this value can change very rapidly and is thus changes are not signalled. Polling it is likely to miss short fluctuations. See MostFull for a better approach to monitoring.

7.1.3.18 ToggleActive()

```
void saftlib::ActionSink::ToggleActive ( )
```

Atomically toggle the active status of conditions.

When reconfiguring an ActionSink, it is sometimes necessary to apply many changes simultaneously. To achieve this, simply use NewCondition to create all the new conditions in the inactive state. Then use this method to simultaneous toggle all conditions on this ActionSink. The new active conditions will be applied such that on one nanosecond, the old set is active and on the next nanosecond the new set is active. Be aware that this function can be used by different applications and it can therefore influence other applications that did not explicitely Own() the ActionSink. To avoid this, always own the ActionSink before using it. Here is the call graph for this function:

7.1.4 Member Data Documentation

7.1.4.1 SigConflict

sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> saftlib::↩
ActionSink::SigConflict

An example of a conflict since last ConflictCount change.

Parameters

count	The new value of ConflictCount when this signal was raised.
event	The event identifier of a conflicting action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The scheduled action execution timestamp (event time + offset).
executed	The timestamp when the action was actually executed.

7.1.4.2 SigDelayed

`sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > saftlib::↩ ActionSink::SigDelayed`

An example of a delayed action the last DelayedCount change.

Parameters

count	The value of DelayedCount when this signal was raised.
event	The event identifier of the delayed action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The desired execution timestamp (event time + offset).
executed	The timestamp when the action was actually executed.

7.1.4.3 SigEarly

`sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> saftlib::↩ ActionSink::SigEarly`

An example of an early action since last EarlyCount change.

Parameters

count	The new value of LateCount when this signal was raised.
event	The event identifier of the early action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The desired execution timestamp (event time + offset).
executed	The actual execution timestamp.

7.1.4.4 SigLate

`sigc::signal<void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> saftlib::↩ ActionSink::SigLate`

: An example of a late action since last LateCount change.

Parameters

count	The new value of LateCount when this signal was raised.
event	The event identifier of the late action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The desired execution timestamp (event time + offset).
executed	The actual execution timestamp. Keep in mind that an action is only counted as late if it is scheduled for the past. An action which leaves the timing receiver after its deadline, due to a slow consumer, is a delayed (not late) action.

The documentation for this class was generated from the following files:

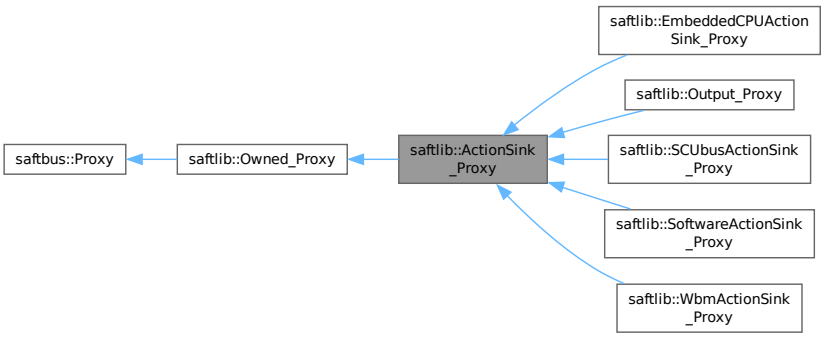
- src/ActionSink.hpp
- src/ActionSink.cpp

7.2 saftlib::ActionSink_Proxy Class Reference

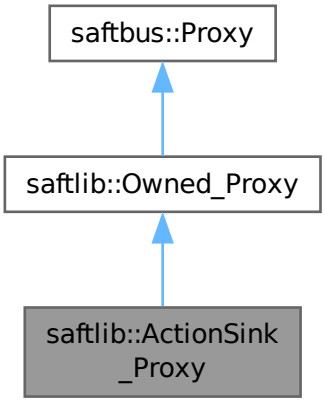
An output through which actions flow.

`#include <ActionSink_Proxy.hpp>`

Inheritance diagram for saftlib::ActionSink_Proxy:



Collaboration diagram for saftlib::ActionSink_Proxy:



Public Member Functions

- **ActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions ()

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions ()

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions ()

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset ()

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset ()

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency ()

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold ()

  Actions further into the future than this are early.
- uint16_t getCapacity ()

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull ()

Report the largest number of pending actions seen.

- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate ()

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount ()

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount ()

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount ()

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount ()

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount ()

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount ()

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< ActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Public Attributes

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.2.1 Detailed Description

An output through which actions flow.

Conditions created on this ActionSink specify which timing events are translated into actions. These actions have execution timestamps which determine when the action is to be executed, precise to the nanosecond.

More specialized versions of this interface provide the 'NewCondition' method to create conditions specific to the type of ActionSink. For example, SoftwareActionSinks create conditions that emit signals to software. This interface captures the functionality common to all ActionSinks, such as atomic toggle, offset constraints, and introspection. In particular, ActionSinks have common failure modes.

Actions are queued for delivery at the appropriate time, in hardware. Hardware has limited storage, reflected by the Fill, Capacity, and MostFull properties. These should be monitored to ensure that the queue never overflows.

The first failure mode of an ActionSink is that the queue overflows. In this case, the hardware drops an action. Obviously, this is a critical error which may result in an undefined state. To prevent these failures, MostFull should be kept below some safety margin with respect to Capacity. Note: several distinct ActionSinks may share underlying hardware, and the Fill property is shared amongst all instances. Each overflow is recorded in the OverflowCount register. Due the rate at which this counter might increase, the API throttles updates using the SignalRate property.

Another critical error for an ActionSink is the possibility of a late action. This indicates that hardware was instructed to execute an action at a time in the past. This is typically caused by either a malfunctioning data master, desynchronized clocks, or conditions with large negative offsets. This is a critical failure as it might leave the system in an undefined state. Conditions may be configured to either drop or execute late actions. If late actions are dropped, a magnet might never be turned off. If late actions are executed, a magnet might be turned on again after it was supposed to be turned off (ie: the actions get misordered). In any case, LateCount is increased.

Similar to late actions, one can also have early actions. If an action is scheduled for execution too far into the future, the timing receiver will choose to mark it as early. This prevents these actions from permanently consuming space in the finite hardware buffers. Early actions are also critical failures as it can leave the system in an undefined state, just as a late action. Conditions may be configured to either drop or execute early actions.

The final misordering failure for an ActionSink is the possibilty of a Conflict. If two actions are scheduled to be executed at the same nanosecond, their relative order is undefined. These conflicts are a critical error as they may leave the system in an undefined state, just as with early and late actions. Conflicts should be prevented by never creating two Conditions on the same ActionSink which could occur at the same time. Note that it is NOT a Conflict for two actions to be executed at the same time by two different ActionSinks. For software, this means that two programs, each with their own SoftwareActionSink do not need to be concerned about conflicts between their schedules. As another example, two different LEMO output cables (corresponding to two OutputActionSinks) can be toggled high at the same time.

Finally, these is the possibility of a delayed action. Unlike late, early, and conflicting actions, delayed actions are never misordered. Thus, delayed actions are typically not as severe a failure mode, and Conditions default to allowing their execution. Delays can happen when the delivery rate of actions (potentially 1GHz) exceeds the capability of the receiver to process the actions. For example, an output might require 100ns to deliver an action. If two actions are scheduled for delivery back-to-back with 1ns between, the second action is delayed. For Software↩ ActionSinks, delays can probably always be ignored because the handler is much much slower than the hardware. For a kicker, on the other hand, a delay is probably a critical error.

7.2.2 Member Function Documentation

7.2.2.1 getActionCount()

```
uint64_t saftlib::ActionSink_Proxy::getActionCount ( )
```

The number of actions processed by the Sink.

Returns

The number of actions processed by the Sink.

As actions can be emitted very rapidly, ActionCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.2.2.2 getActiveConditions()

```
std::vector< std::string > saftlib::ActionSink_Proxy::getActiveConditions ( )
```

All active conditions created on this ActionSink.

Returns

All active conditions created on this ActionSink.

All active conditions on the ActionSink. Each object path implements the the matching Condition interface; for example, a SoftwareActionSink will have SoftwareConditions.

7.2.2.3 getAllConditions()

```
std::vector< std::string > saftlib::ActionSink_Proxy::getAllConditions ( )
```

All conditions created on this ActionSink.

Returns

All conditions created on this ActionSink.

All active and inactive conditions on the ActionSink. Each object path implements the the matching Condition interface; for example, a SoftwareActionSink will have SoftwareConditions.

7.2.2.4 getCapacity()

```
uint16_t saftlib::ActionSink_Proxy::getCapacity ( )
```

The maximum number of actions queueable without Overflow.

Returns

The maximum number of actions queueable without Overflow.

The timing receiver hardware can only queue a limited number of actions. This property reports the maximum number of actions that may be simultaneously queued. Be aware that this resource may be shared between multiple ActionSinks. For example, all SoftwareActionSinks share a common underlying queue in hardware. This Capacity represents the total size, which must be shared.

7.2.2.5 getConflictCount()

```
uint64_t saftlib::ActionSink_Proxy::getConflictCount ( )
```

The number of actions which conflicted.

Returns

The number of actions which conflicted.

If two actions should have been executed simultaneously by the same ActionSink, they are executed in an undefined order. Each time this happens, the ConflictCount is increased. As conflicts can occur very rapidly, ConflictCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.2.2.6 getDelayedCount()

```
uint64_t saftlib::ActionSink_Proxy::getDelayedCount ( )
```

The number of actions which have been delayed.

Returns

The number of actions which have been delayed.

The timing receiver emits actions potentially every nanosecond. In the case that the receiver cannot immediately process an action, the timing receiver delays the action until the receiver is ready. This can happen either because the receiver was still busy with a previous action or the output was used externally (bus arbitration). As actions can be emitted very rapidly, DelayedCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.2.2.7 getEarlyCount()

```
uint64_t saftlib::ActionSink_Proxy::getEarlyCount ( )
```

The number of actions delivered early.

Returns

The number of actions delivered early.

As described in the interface overview, an action can be early due to a buggy data master, loss of clock synchronization, or very positive condition offsets. This is a critical failure as it can result in misordering of executed actions. Each such failure increases this counter.

As early acitons can occur very rapidly, EarlyCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.2.2.8 getEarlyThreshold()

```
uint64_t saftlib::ActionSink_Proxy::getEarlyThreshold ( )
```

Actions further into the future than this are early.

Returns

Actions further into the future than this are early.

If an action is scheduled for execution too far into the future, it gets truncated to at most EarlyThreshold nanoseconds into the future.

7.2.2.9 getInactiveConditions()

```
std::vector< std::string > saftlib::ActionSink_Proxy::getInactiveConditions ( )
```

All inactive conditions created on this ActionSink.

Returns

All inactive conditions created on this ActionSink.

All inactive conditions on the ActionSink. Each object path implements the the matching Condition interface; for example, a SoftwareActionSink will have SoftwareConditions.

7.2.2.10 getLateCount()

```
uint64_t saftlib::ActionSink_Proxy::getLateCount ( )
```

The number of actions delivered late.

Returns

The number of actions delivered late.

As described in the interface overview, an action can be late due to a buggy data master, loss of clock synchronization, or very negative condition offsets. This is a critical failure as it can result in misordering of executed actions. Each such failure increases this counter.

As late actions can occur very rapidly, LateCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.2.2.11 getLatency()

```
uint64_t saftlib::ActionSink_Proxy::getLatency ( )
```

Nanoseconds between event and earliest execution of an action.

Returns

Nanoseconds between event and earliest execution of an action.

7.2.2.12 getMaxOffset()

```
int64_t saftlib::ActionSink_Proxy::getMaxOffset ( )
```

Maximum allowed offset (nanoseconds) usable in NewCondition.

Returns

Maximum allowed offset (nanoseconds) usable in NewCondition.

Large offsets are almost always an error. A large positive offset could result in early actions being created. By default, no condition may have an offset larger than 1s. Attempts to create conditions with offsets greater than MaxOffset result in an error. Change this property to override this safety feature.

7.2.2.13 getMinOffset()

```
int64_t saftlib::ActionSink_Proxy::getMinOffset ( )
```

Minimum allowed offset (nanoseconds) usable in NewCondition.

Returns

Minimum allowed offset (nanoseconds) usable in NewCondition.

Large offsets are almost always an error. A very negative offset could result in Late actions. By default, no condition may be created with an offset smaller than -100us. Attempts to create conditions with offsets less than MinOffset result in an error. Change this property to override this safety feature.

7.2.2.14 getMostFull()

```
uint16_t saftlib::ActionSink_Proxy::getMostFull ( )
```

Report the largest number of pending actions seen.

Returns

Report the largest number of pending actions seen.

The timing receiver hardware can only queue a limited number of actions. This property reports the highest Fill level seen by the hardware since it was last reset to 0. Keep in mind that the queue may be shared, including actions from all users of the underlying hardware. So two programs each using a SoftwareActionSink will potentially see an increase in this value when both programs are active.

7.2.2.15 getOverflowCount()

```
uint64_t saftlib::ActionSink_Proxy::getOverflowCount ( )
```

The number of actions lost due to Overflow.

The underlying hardware queue may overflow once Fill=Capacity. This is a critical error condition that must be handlded. The causes may be either: 1- the actions have an execution time far enough in the future that too many actions are buffered before they are executed, or 2- the receiving component is unable to execute actions as quickly as the timing system delivers them. The second case is particularly likely for SoftwareActionSinks that attempt to listen to high frequency events. Even though SoftwareActionSinks share a common queue, Overflow is reported only to the ActionSink whose action was dropped. As overflows can occur very rapidly, OverflowCount may increase by more than 1 between emissions. There is a minimum delay of SignalRate nanoseconds between updates to this property.

7.2.2.16 getSignalRate()

```
uint64_t saftlib::ActionSink_Proxy::getSignalRate ( )
```

Minimum interval between updates (nanoseconds, default 100ms).

Returns

Minimum interval between updates (nanoseconds, default 100ms).

The properties OverflowCount, ActionCount, LateCount, EarlyCount, ConflictCount, and DelayedCount can increase rapidly. To prevent excessive CPU load, SignalRate imposes a minimum cooldown between updates to these values.

7.2.2.17 ReadFill()

```
uint16_t saftlib::ActionSink_Proxy::ReadFill ( )
```

Report the number of currently pending actions.

Returns

Number of pending actions.

The timing receiver hardware can only queue a limited number of actions. This method reports the current number of queued actions, which includes actions from all users of the underlying hardware. So two programs each using a SoftwareActionSink will potentially see an increase in this value when both programs are active. Also, this value can change very rapidly and is thus changes are not signalled. Polling it is likely to miss short fluctuations. See MostFull for a better approach to monitoring.

7.2.2.18 signal_dispatch()

```
bool saftlib::ActionSink_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

Reimplemented in saftlib::EmbeddedCPUActionSink_Proxy, saftlib::Output_Proxy, saftlib::SCUbusActionSink_Proxy, saftlib::SoftwareActionSink_Proxy, and saftlib::WbmActionSink_Proxy.

7.2.2.19 ToggleActive()

```
void saftlib::ActionSink_Proxy::ToggleActive ( )
```

Atomically toggle the active status of conditions.

When reconfiguring an ActionSink, it is sometimes necessary to apply many changes simultaneously. To achieve this, simply use NewCondition to create all the new conditions in the inactive state. Then use this method to simultaneous toggle all conditions on this ActionSink. The new active conditions will be applied such that on one nanosecond, the old set is active and on the next nanosecond the new set is active. Be aware that this function can be used by different applications and it can therefore influence other applications that did not explicitely Own() the ActionSink. To avoid this, always own the ActionSink before using it.

7.2.3 Member Data Documentation

7.2.3.1 SigConflict

```
sigc::signal<void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> saftlib::↩
ActionSink_Proxy::SigConflict
```

An example of a conflict since last ConflictCount change.

Parameters

count	The new value of ConflictCount when this signal was raised.
event	The event identifier of a conflicting action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The scheduled action execution timestamp (event time + offset).
executed	The timestamp when the action was actually executed.

7.2.3.2 SigDelayed

```
sigc::signal<void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> saftlib::↩
ActionSink_Proxy::SigDelayed
```

An example of a delayed action the last DelayedCount change.

Parameters

count	The value of DelayedCount when this signal was raised.
event	The event identifier of the delayed action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The desired execution timestamp (event time + offset).
executed	The timestamp when the action was actually executed.

7.2.3.3 SigEarly

```
sigc::signal<void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> saftlib::↩
ActionSink_Proxy::SigEarly
```

An example of an early action since last EarlyCount change.

Parameters

count	The new value of LateCount when this signal was raised.
event	The event identifier of the early action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The desired execution timestamp (event time + offset).
executed	The actual execution timestamp.

7.2.3.4 SigLate

```
sigc::signal<void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> saftlib::↩
ActionSink_Proxy::SigLate
```

: An example of a late action since last LateCount change.

Parameters

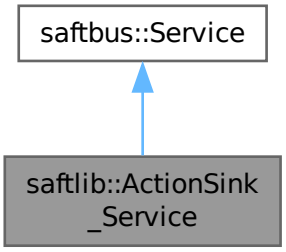
count	The new value of LateCount when this signal was raised.
event	The event identifier of the late action.
param	The parameter field, whose meaning depends on the event ID.
deadline	The desired execution timestamp (event time + offset).
executed	The actual execution timestamp. Keep in mind that an action is only counted as late if it is scheduled for the past. An action which leaves the timing receiver after its deadline, due to a slow consumer, is a delayed (not late) action.

The documentation for this class was generated from the following files:

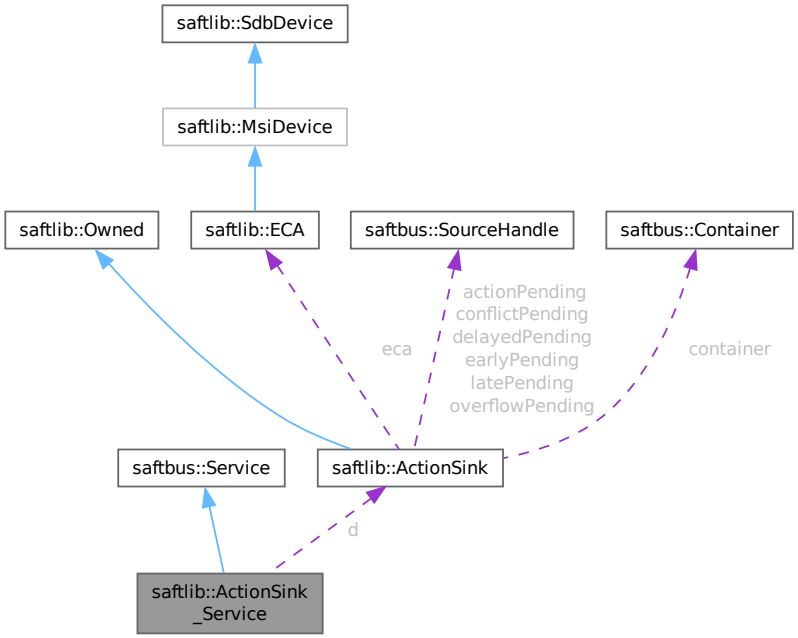
- src/ActionSink_Proxy.hpp
- src/ActionSink_Proxy.cpp

7.3 saftlib::ActionSink_Service Class Reference

Inheritance diagram for saftlib::ActionSink_Service:



Collaboration diagram for saftlib::ActionSink_Service:



Public Types

- typedef ActionSink **DriverType**

Public Member Functions

- **ActionSink_Service** (ActionSink *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **OverflowCount_dispatch_function** (uint64_t arg_1)
- void **ActionCount_dispatch_function** (uint64_t arg_1)
- void **LateCount_dispatch_function** (uint64_t arg_1)
- void **SigLate_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **EarlyCount_dispatch_function** (uint64_t arg_1)
- void **SigEarly_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **ConflictCount_dispatch_function** (uint64_t arg_1)
- void **SigConflict_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **DelayedCount_dispatch_function** (uint64_t arg_1)
- void **SigDelayed_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- ActionSink * **d**
- sigc::connection **OverflowCount_connection**
- sigc::connection **ActionCount_connection**
- sigc::connection **LateCount_connection**
- sigc::connection **SigLate_connection**
- sigc::connection **EarlyCount_connection**
- sigc::connection **SigEarly_connection**
- sigc::connection **ConflictCount_connection**
- sigc::connection **SigConflict_connection**
- sigc::connection **DelayedCount_connection**
- sigc::connection **SigDelayed_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.3.1 Member Function Documentation

7.3.1.1 call()

```
void saftlib::ActionSink_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/ActionSink_Service.hpp
- src/ActionSink_Service.cpp

7.4 Allocator Class Reference

`#include <global_allocator.hpp>`

Public Member Functions

- char * **malloc** (size_t n)
- void **free** (char *ptr)

7.4.1 Detailed Description

Author

Michael Reese m.reese@gsi.de

The documentation for this class was generated from the following file:

- saftbus/global_allocator.hpp

7.5 saftbus::Allocator Class Reference

Public Member Functions

- char * **malloc** (size_t n)
- void **free** (char *ptr)
- std::string **fillstate** ()

The documentation for this class was generated from the following files:

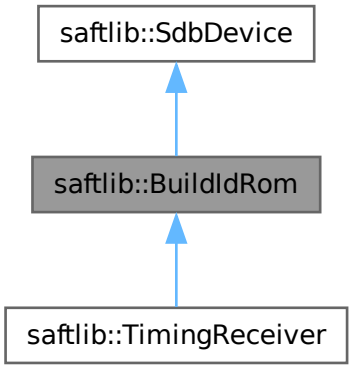
- saftbus/configurable_chunck_allocator_rt.hpp
- saftbus/configurable_chunck_allocator_rt.cpp

7.6 saftlib::BuildIdRom Class Reference

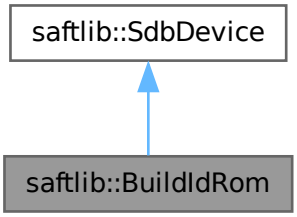
Representation of the SDB device with build id information. It can be used to obtain strings with gateware info and version numbers.

`#include <BuildIdRom.hpp>`

Inheritance diagram for saftlib::BuildIdRom:



Collaboration diagram for saftlib::BuildIdRom:



Public Member Functions

- **BuildIdRom** (etherbone::Device &device)
- std::map< std::string, std::string > getGatewareInfo () const

  Key-value map of hardware build information.
- std::string getGatewareVersion () const

  Hardware build version.

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.6.1 Detailed Description

Representation of the SDB device with build id information. It can be used to obtain strings with gateware info and version numbers.

7.6.2 Member Function Documentation

7.6.2.1 getGatewareInfo()

```
std::map< std::string, std::string > saftlib::BuildIdRom::getGatewareInfo ( ) const
```

Key-value map of hardware build information.

Returns

Key-value map of hardware build information

7.6.2.2 getGatewareVersion()

```
std::string saftlib::BuildIdRom::getGatewareVersion ( ) const
```

Hardware build version.

Returns

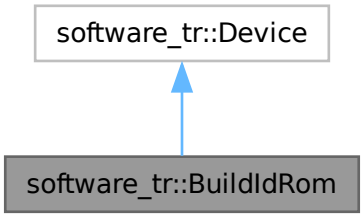
¨major.minor.tiny¨ if version is valid (or ¨N/A¨ if not available)

The documentation for this class was generated from the following files:

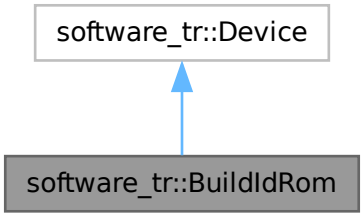
- src/BuildIdRom.hpp
- src/BuildIdRom.cpp

7.7 software_tr::BuildIdRom Class Reference

Inheritance diagram for software_tr::BuildIdRom:



Collaboration diagram for software_tr::BuildIdRom:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0x2d39fa8b }

Public Member Functions

- **BuildIdRom** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool read_access (uint32_t adr, int sel, uint32_t *dat_out)
- bool write_access (uint32_t adr, int sel, uint32_t dat)

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.7.1 Member Function Documentation

7.7.1.1 contains()

```
bool software_tr::BuildIdRom::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.7.1.2 read_access()

```
bool software_tr::BuildIdRom::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

7.7.1.3 write_access()

```
bool software_tr::BuildIdRom::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

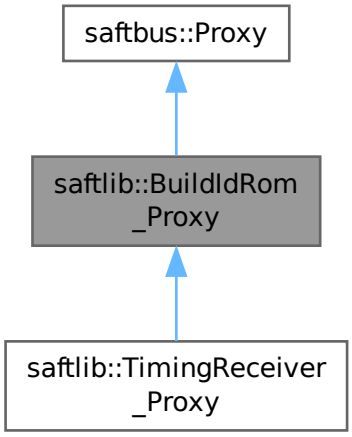
- src/saft-software-tr.cpp

7.8 saftlib::BuildIdRom_Proxy Class Reference

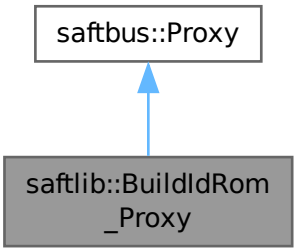
Representation of the SDB device with build id information. It can be used to obtain strings with gateware info and version numbers.

```
#include <BuildIdRom_Proxy.hpp>
```

Inheritance diagram for saftlib::BuildIdRom_Proxy:



Collaboration diagram for saftlib::BuildIdRom_Proxy:



Public Member Functions

- **BuildIdRom_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::map< std::string, std::string > getGatewareInfo ()

  Key-value map of hardware build information.
- std::string getGatewareVersion ()

  Hardware build version.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< BuildIdRom_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.8.1 Detailed Description

Representation of the SDB device with build id information. It can be used to obtain strings with gateware info and version numbers.

7.8.2 Member Function Documentation

7.8.2.1 getGatewareInfo()

```
std::map< std::string, std::string > saftlib::BuildIdRom_Proxy::getGatewareInfo ( )
```

Key-value map of hardware build information.

Returns

Key-value map of hardware build information

7.8.2.2 getGatewareVersion()

```
std::string saftlib::BuildIdRom_Proxy::getGatewareVersion ( )
```

Hardware build version.

Returns

¨major.minor.tiny¨ if version is valid (or ¨N/A¨ if not available)

7.8.2.3 signal_dispatch()

```
bool saftlib::BuildIdRom_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

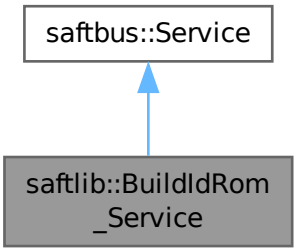
Reimplemented in saftlib::TimingReceiver_Proxy.

The documentation for this class was generated from the following files:

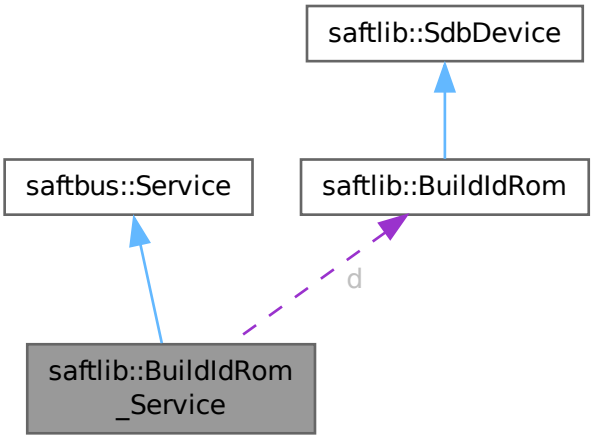
- src/BuildIdRom_Proxy.hpp
- src/BuildIdRom_Proxy.cpp

7.9 saftlib::BuildIdRom_Service Class Reference

Inheritance diagram for saftlib::BuildIdRom_Service:



Collaboration diagram for saftlib::BuildIdRom_Service:



Public Types

- typedef BuildIdRom **DriverType**

Public Member Functions

- **BuildIdRom_Service** (BuildIdRom *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- BuildIdRom * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.9.1 Member Function Documentation

7.9.1.1 call()

```
void saftlib::BuildIdRom_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

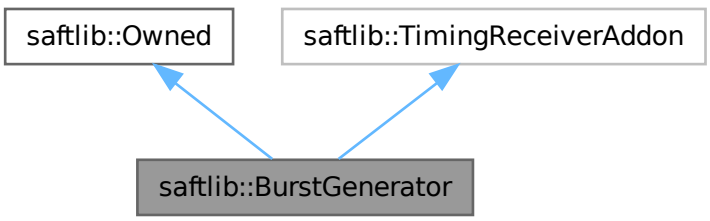
- src/BuildIdRom_Service.hpp
- src/BuildIdRom_Service.cpp

7.10 saftlib::BurstGenerator Class Reference

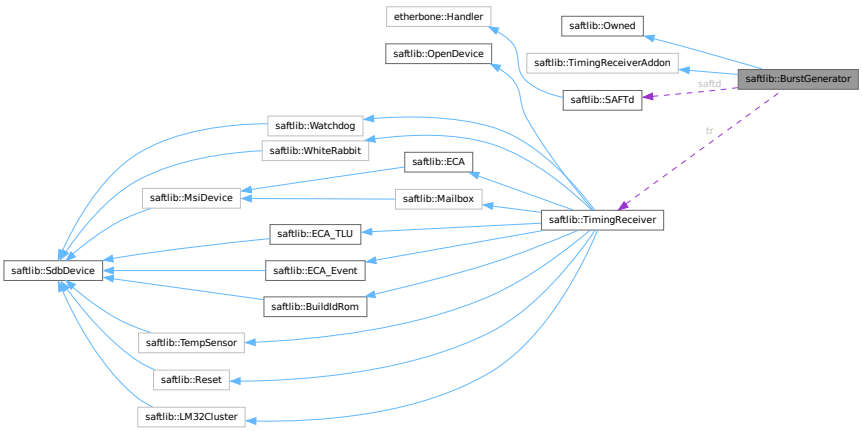
Burst generation service.

`#include <BurstGenerator.hpp>`

Inheritance diagram for saftlib::BurstGenerator:



Collaboration diagram for saftlib::BurstGenerator:



Public Member Functions

- std::map< std::string, std::map< std::string, std::string > > getObjects ()
- std::string **getObjectPath** ()
- **BurstGenerator** (saftbus::Container *container, SAFTd *saft_daemon, TimingReceiver *timing_receiver)
- int32_t instruct (uint32_t code, const std::vector< uint32_t > &args)

  Instruction to the burst generator.
- std::vector< uint32_t > readBurstInfo (uint32_t id)

  Get burst info.
- std::vector< uint32_t > readSharedBuffer (uint32_t size)

  Read the shared memory.
- uint32_t readState ()

  Read the actual state of the burst generator.
- uint32_t getResponse () const

  The burst generator response sent via mailbox.

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

- virtual std::map< std::string, std::map< std::string, std::string > > **getObjects** ()=0

Public Attributes

- sigc::signal< void, uint32_t > sigInstComplete

  Notify the completion of an instruction.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions

- bool **firmwareRunning** (uint32_t id)
- void **msi_handler** (eb_data_t msg)

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes

- std::string **objectPath**
- SAFTd * **saftd**
- TimingReceiver * **tr**
- etherbone::Device & **device**
- std::unique_ptr< IRQ > **my_msi**
- std::unique_ptr< Mailbox::Slot > **my_slot**
- int **bg_slot**
- uint32_t **response**
- bool **found_bg_fw**
- eb_address_t **ram_base**
- std::vector< eb_address_t > **shm_buffer**

7.10.1 Detailed Description

Burst generation service.

A dedicated firmware must be loaded and running in the embedded LM32 core prior to the burst generation. The methods and signals of this interface are responsible to inform about the underlying hardware, firmware and load the firmware binary if necessary.

7.10.2 Member Function Documentation

7.10.2.1 getObjects()

```
std::map< std::string, std::map< std::string, std::string > > saftlib::BurstGenerator::getObjects ( ) [virtual]
```

Implements saftlib::TimingReceiverAddon.

7.10.2.2 getResponse()

```
uint32_t saftlib::BurstGenerator::getResponse ( ) const
```

The burst generator response sent via mailbox.

Parameters

code	User instruction code

7.10.2.3 instruct()

```
int32_t saftlib::BurstGenerator::instruct (
            uint32_t code,
            const std::vector< uint32_t > & args )
```

Instruction to the burst generator.

It is a common method to communicate with the burst generator. The method contains an instruction code and corresponding arguments.

Parameters

code	User instruction code for LM32
args	Instruction arguments (vector of u32 integers)
result	Return result. 0 on success.

7.10.2.4 readBurstInfo()

```
std::vector< uint32_t > saftlib::BurstGenerator::readBurstInfo (
            uint32_t id )
```

Get burst info.

The info includes the type and index of IO port, IDs of trigger and toggling events.

Parameters

id	The burst ID
info	The burst info (vector of u32 integers)

7.10.2.5 readSharedBuffer()

```
std::vector< uint32_t > saftlib::BurstGenerator::readSharedBuffer (
            uint32_t size )
```

Read the shared memory.

Parameters

size	The amount of data to be read
data	The read data

7.10.2.6 readState()

```
uint32_t saftlib::BurstGenerator::readState ( )
```

Read the actual state of the burst generator.

Parameters

state	Actual state of the burst generator

7.10.3 Member Data Documentation

7.10.3.1 sigInstComplete

```
sigc::signal<void, uint32_t> saftlib::BurstGenerator::sigInstComplete
```

Notify the completion of an instruction.

It is the response of the burst generator on user instruction request.

Parameters

code	User instruction code

The documentation for this class was generated from the following files:

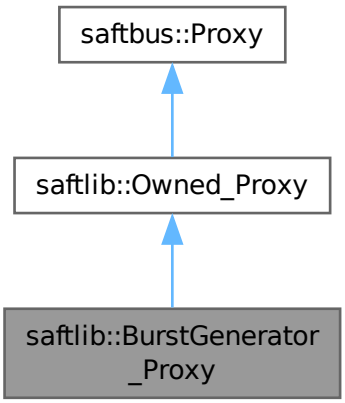
- src/BurstGenerator.hpp
- src/BurstGenerator.cpp

7.11 saftlib::BurstGenerator_Proxy Class Reference

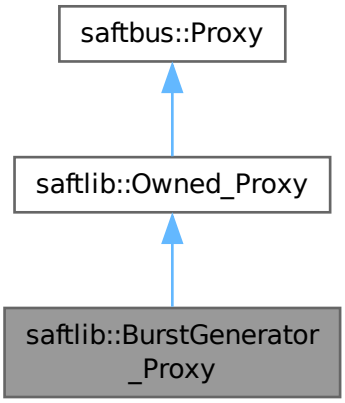
Burst generation service.

```
#include <BurstGenerator_Proxy.hpp>
```

Inheritance diagram for saftlib::BurstGenerator_Proxy:



Collaboration diagram for saftlib::BurstGenerator_Proxy:



Public Member Functions

- **BurstGenerator_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::↩ SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- int32_t instruct (uint32_t code, const std::vector< uint32_t > &args)

  Instruction to the burst generator.
- std::vector< uint32_t > readBurstInfo (uint32_t id)

Get burst info.

- std::vector< uint32_t > readSharedBuffer (uint32_t size)

  Read the shared memory.
- uint32_t readState ()

  Read the actual state of the burst generator.
- uint32_t getResponse ()

  The burst generator response sent via mailbox.

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< BurstGenerator_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Public Attributes

- sigc::signal< void, uint32_t > sigInstComplete

  Notify the completion of an instruction.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.11.1 Detailed Description

Burst generation service.

A dedicated firmware must be loaded and running in the embedded LM32 core prior to the burst generation. The methods and signals of this interface are responsible to inform about the underlying hardware, firmware and load the firmware binary if necessary.

7.11.2 Member Function Documentation

7.11.2.1 getResponse()

```
uint32_t saftlib::BurstGenerator_Proxy::getResponse ( )
```

The burst generator response sent via mailbox.

Parameters

code	User instruction code

7.11.2.2 instruct()

```
int32_t saftlib::BurstGenerator_Proxy::instruct (
            uint32_t code,
            const std::vector< uint32_t > & args )
```

Instruction to the burst generator.

It is a common method to communicate with the burst generator. The method contains an instruction code and corresponding arguments.

Parameters

code	User instruction code for LM32
args	Instruction arguments (vector of u32 integers)
result	Return result. 0 on success.

7.11.2.3 readBurstInfo()

```
std::vector< uint32_t > saftlib::BurstGenerator_Proxy::readBurstInfo (
            uint32_t id )
```

Get burst info.

The info includes the type and index of IO port, IDs of trigger and toggling events.

Parameters

id	The burst ID
info	The burst info (vector of u32 integers)

7.11.2.4 readSharedBuffer()

```
std::vector< uint32_t > saftlib::BurstGenerator_Proxy::readSharedBuffer (
            uint32_t size )
```

Read the shared memory.

Parameters

size	The amount of data to be read
data	The read data

7.11.2.5 readState()

```
uint32_t saftlib::BurstGenerator_Proxy::readState ( )
```

Read the actual state of the burst generator.

Parameters

state	Actual state of the burst generator

7.11.2.6 signal_dispatch()

```
bool saftlib::BurstGenerator_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

7.11.3 Member Data Documentation

7.11.3.1 sigInstComplete

```
sigc::signal<void, uint32_t> saftlib::BurstGenerator_Proxy::sigInstComplete
```

Notify the completion of an instruction.

It is the response of the burst generator on user instruction request.

Parameters

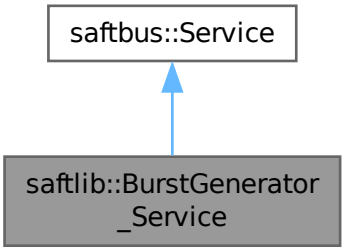
code	User instruction code

The documentation for this class was generated from the following files:

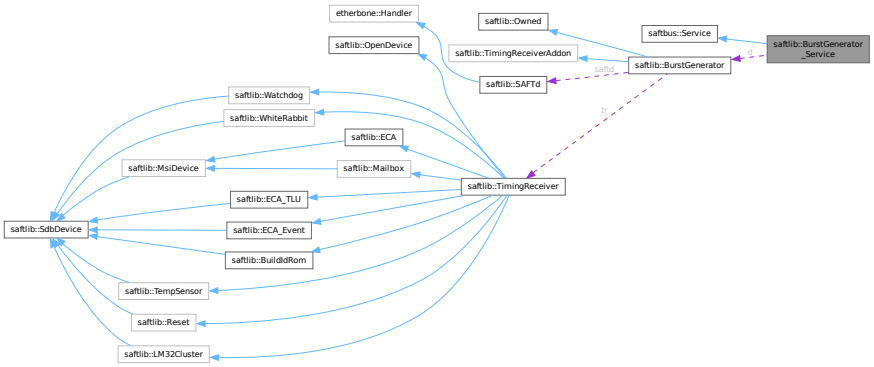
- src/BurstGenerator_Proxy.hpp
- src/BurstGenerator_Proxy.cpp

7.12 saftlib::BurstGenerator_Service Class Reference

Inheritance diagram for saftlib::BurstGenerator_Service:



Collaboration diagram for saftlib::BurstGenerator_Service:



Public Types

- typedef BurstGenerator **DriverType**

Public Member Functions

- **BurstGenerator_Service** (BurstGenerator *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **sigInstComplete_dispatch_function** (uint32_t arg_1)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- BurstGenerator * **d**
- sigc::connection **sigInstComplete_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.12.1 Member Function Documentation

7.12.1.1 call()

```
void saftlib::BurstGenerator_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/BurstGenerator_Service.hpp
- src/BurstGenerator_Service.cpp

7.13 ChunckAllocatorRT< MAX_CHUNCKS, CHUNCKSIZE > Class Template Reference

`#include <chunck_allocator_rt.hpp>`

Public Member Functions

- char * **malloc** (size_t size)
- void **free** (char *ptr)
- void **print_size** ()
- void **print_state** ()
- bool **contains** (char *ptr)
- bool **full** ()
- bool **fits** (size_t n)

7.13.1 Detailed Description

template<size_t MAX_CHUNCKS, size_t CHUNCKSIZE>
class ChunckAllocatorRT< MAX_CHUNCKS, CHUNCKSIZE >

Author

Michael Reese m.reese@gsi.de

The documentation for this class was generated from the following file:

- saftbus/chunck_allocator_rt.hpp

7.14 saftbus::ChunckAllocatorRT Class Reference

Public Member Functions

- **ChunckAllocatorRT** (size_t max_chuncks, size_t chuncksize)
- char ∗ **malloc** (size_t size)
- void **free** (char ∗ptr)
- void **print_size** ()
- void **print_state** ()
- bool **contains** (char ∗ptr)
- bool **full** ()
- bool **fits** (size_t n)

Friends

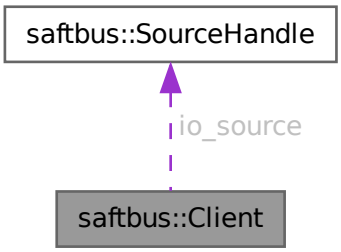
- class **Allocator**

The documentation for this class was generated from the following files:

- saftbus/configurable_chunck_allocator_rt.hpp
- saftbus/configurable_chunck_allocator_rt.cpp

7.15 saftbus::Client Struct Reference

Collaboration diagram for saftbus::Client:



Public Member Functions

- **Client** (int fd, pid_t pid, SourceHandle h)
- void **use_signal_fd** (int fd)
- void **release_signal_fd** (int fd)

Public Attributes

- int **socket_fd**
- pid_t **process_id**
- SourceHandle **io_source**
- std::map< int, int > **signal_fd_use_count**

The documentation for this struct was generated from the following file:

- saftbus/server.cpp

7.16 saftbus::ClientConnection Class Reference

Establish a connection to a running saftbus::ServerConnection using a named socket (¨/var/run/saftbus/saftbus¨ by default)

```
#include <client.hpp>
```

Classes

- struct Impl

Public Member Functions

- **ClientConnection** (const std::string &socket_name=¨/var/run/saftbus/saftbus¨)
- int send (Serializer &serializer, int timeout_ms=-1)

  send whatever data is in serial buffer to the server
- int receive (Deserializer &deserializer, int timeout_ms=-1)

  wait for data to arrive from the server
- int atomic_send_and_receive (Serializer &serializer, Deserializer &deserializer, int timeout_ms=-1)

  call send and receiver atomically

Friends

- class **SignalGroup**
- class **Proxy**

7.16.1 Detailed Description

Establish a connection to a running saftbus::ServerConnection using a named socket (¨/var/run/saftbus/saftbus¨ by default)

A connection is established by creating a socket pair (using socket_pair function) and send one of the file descriptors to the ServerConnection while keeping the other file descriptor open. This established an exclusive communication channel between the process with the ClientConnection an the process with the ServerConnection. The ServerConnection manages multiple such communication channels.

Parameters

socket_name	the location of the named socket from the saftbus::ServerConnection

7.16.2 Member Function Documentation

7.16.2.1 atomic_send_and_receive()

```
int saftbus::ClientConnection::atomic_send_and_receive (
            Serializer & serializer,
            Deserializer & deserializer,
            int timeout_ms = -1 )
```

call send and receiver atomically

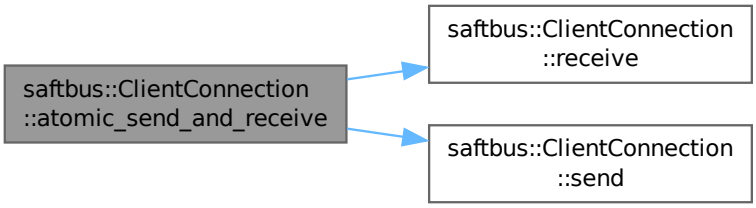
Parameters

serializer	should contain serialized data
deserializer	that contains the received buffer after the function returns
timeout	return after so many milliseconds even if the data could not be sent.

Returns

0 in case of timeout, >0 in case of success, -1 in case of error

Here is the call graph for this function:



7.16.2.2 receive()

```
int saftbus::ClientConnection::receive (
            Deserializer & deserializer,
            int timeout_ms = -1 )
```

wait for data to arrive from the server

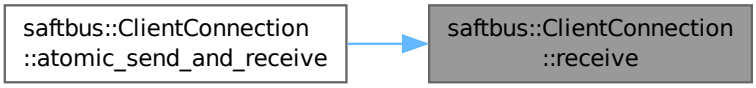
Parameters

deserializer	that contains the received buffer after the function returns
timeout	return after so many milliseconds even if the data could not be sent.

Returns

0 in case of timeout, >0 in case of success, -1 in case of error

Here is the caller graph for this function:



7.16.2.3 send()

```
int saftbus::ClientConnection::send (
            Serializer & serializer,
            int timeout_ms = -1 )
```

send whatever data is in serial buffer to the server

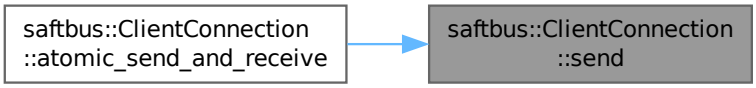
Parameters

serializer	should contain serialized data
timeout	return after so many milliseconds even if the data could not be sent.

Returns

0 in case of timeout, >0 in case of success, -1 in case of error

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

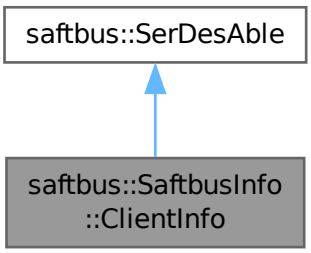
- saftbus/client.hpp
- saftbus/client.cpp

7.17 saftbus::SaftbusInfo::ClientInfo Struct Reference

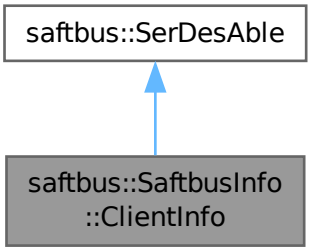
contains all information about a saftbus client connected to a saftbus server

`#include <client.hpp>`

Inheritance diagram for saftbus::SaftbusInfo::ClientInfo:



Collaboration diagram for saftbus::SaftbusInfo::ClientInfo:



Public Member Functions

- void serialize (Serializer &ser) const

  custom serializer
- void deserialize (const Deserializer &des)

  custom deserializer

- virtual void serialize (Serializer &ser) const =0
- virtual void deserialize (const Deserializer &des)=0

Public Attributes

- pid_t **process_id**
- int **client_fd**
- std::map< int, int > **signal_fds**

7.17.1 Detailed Description

contains all information about a saftbus client connected to a saftbus server

7.17.2 Member Function Documentation

7.17.2.1 deserialize()

```
void saftbus::SaftbusInfo::ClientInfo::deserialize (
            const Deserializer & des )  [inline], [virtual]
```

custom deserializer

Implements saftbus::SerDesAble.

7.17.2.2 serialize()

```
void saftbus::SaftbusInfo::ClientInfo::serialize (
            Serializer & ser ) const  [inline], [virtual]
```

custom serializer

Implements saftbus::SerDesAble.

The documentation for this struct was generated from the following file:

- saftbus/client.hpp

7.18 saftbus::ServerConnection::ClientInfo Struct Reference

Public Attributes

- pid_t **process_id**
- int **client_fd**
- std::map< int, int > **signal_fds**

The documentation for this struct was generated from the following file:

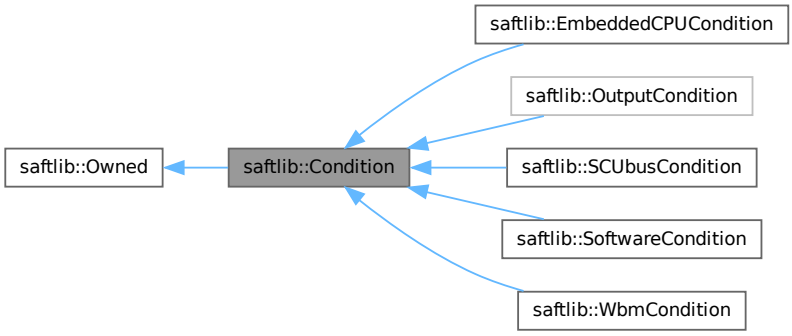
- saftbus/server.hpp

7.19 saftlib::Condition Class Reference

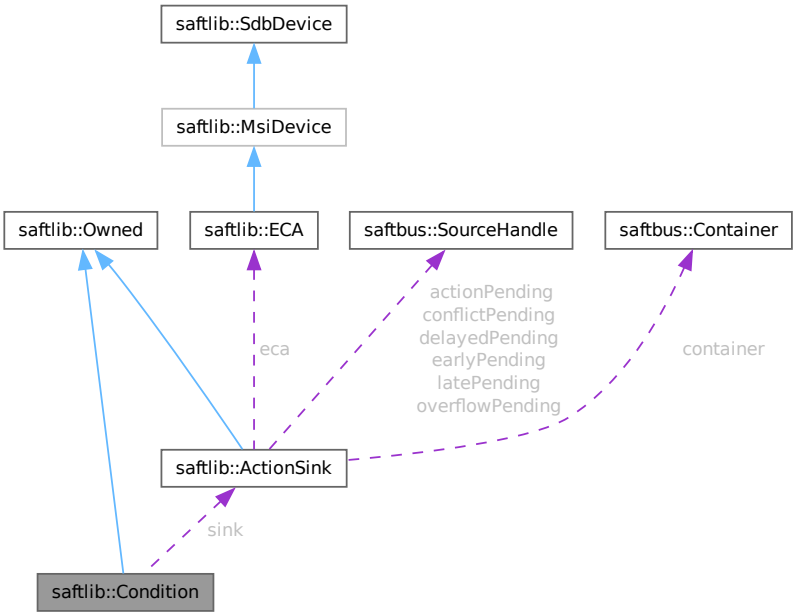
A rule matched against incoming events.

`#include <Condition.hpp>`

Inheritance diagram for saftlib::Condition:



Collaboration diagram for saftlib::Condition:



Public Member Functions

- **Condition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)

- uint64_t getID () const

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask () const

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset () const

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate () const

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly () const

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict () const

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed () const

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive () const

  The condition should be actively matched against events.
- void **setActive** (bool val)
- std::string & **getObjectPath** ()
- uint32_t **getRawTag** () const
- void **setRawActive** (bool val)
- unsigned **getNumber** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Protected Attributes

- std::string **objectPath**
- ActionSink * **sink**
- unsigned **number**
- uint64_t **id**
- uint64_t **mask**
- int64_t **offset**
- uint32_t **tag**
- bool **acceptLate**
- bool **acceptEarly**
- bool **acceptConflict**
- bool **acceptDelayed**
- bool **active**

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed
  The object was destroyed.

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const
  Throw an exception if the caller is not the owner.

7.19.1 Detailed Description

A rule matched against incoming events.

de.gsi.saftlib.Condition:

Conditions are created for ActionSinks to select which events the sink should respond to. Different ActionSinks return handles to different Conditions. This interface is the common denominator.

7.19.2 Member Function Documentation

7.19.2.1 getAcceptConflict()

```
bool saftlib::Condition::getAcceptConflict ( ) const
```

Should conflicting actions be executed? Defaults to false.

Returns

Should conflicting actions be executed? Defaults to false

7.19.2.2 getAcceptDelayed()

```
bool saftlib::Condition::getAcceptDelayed ( ) const
```

Should delayed actions be executed? Defaults to true.

Returns

Should delayed actions be executed? Defaults to true

7.19.2.3 getAcceptEarly()

```
bool saftlib::Condition::getAcceptEarly ( ) const
```

Should early actions be executed? Defaults to false.

Returns

Should early actions be executed? Defaults to false

7.19.2.4 getAcceptLate()

```
bool saftlib::Condition::getAcceptLate ( ) const
```

Should late actions be executed? Defaults to false -->

Returns

Should late actions be executed? Defaults to false -->

7.19.2.5 getActive()

```
bool saftlib::Condition::getActive ( ) const
```

The condition should be actively matched against events.

Returns

The condition should be actively matched against events.

An inactive condition is not used to match against events. You can toggle the active state of this condition via this property, or if multiple Conditions should be atomically adjusted, use the ToggleActive method on the ActionSink.

7.19.2.6 getID()

```
uint64_t saftlib::Condition::getID ( ) const
```

The event identifier which this condition matches against.

Returns

The event identifier which this condition matches against.

An incoming event matches if its ID and this ID agree on all the bits set in this condition's Mask.

7.19.2.7 getMask()

```
uint64_t saftlib::Condition::getMask ( ) const
```

The mask used when comparing event IDs.

Returns

The mask used when comparing event IDs.

An incoming event matches if its ID and the ID property of this Condition agree on all the bits set in this Mask.

7.19.2.8 getOffset()

```
int64_t saftlib::Condition::getOffset ( ) const
```

Added to an event's time to calculate the action's time.

Returns

Added to an event's time to calculate the action's time.

The documentation for this class was generated from the following files:

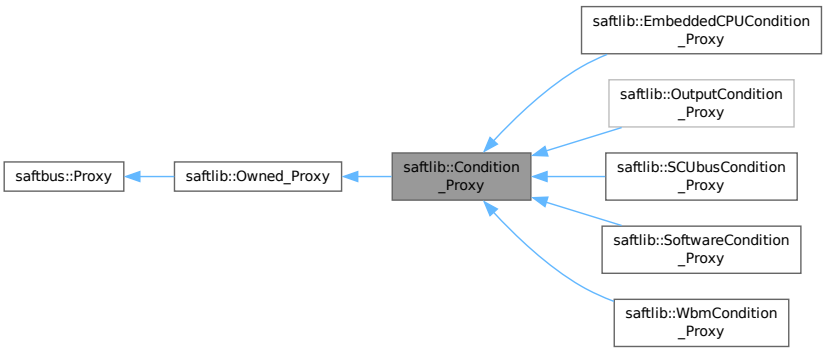
- src/Condition.hpp
- src/Condition.cpp

7.20 saftlib::Condition_Proxy Class Reference

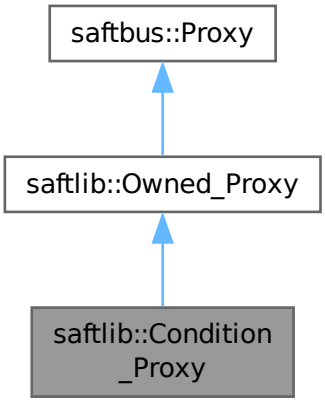
A rule matched against incoming events.

`#include <Condition_Proxy.hpp>`

Inheritance diagram for saftlib::Condition_Proxy:



Collaboration diagram for saftlib::Condition_Proxy:



Public Member Functions

- **Condition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t getID ()

The event identifier which this condition matches against.

- void **setID** (uint64_t val)
- uint64_t getMask ()

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset ()

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate ()

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly ()

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict ()

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed ()

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive ()

  The condition should be actively matched against events.
- void **setActive** (bool val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< Condition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.20.1 Detailed Description

A rule matched against incoming events.

de.gsi.saftlib.Condition:

Conditions are created for ActionSinks to select which events the sink should respond to. Different ActionSinks return handles to different Conditions. This interface is the common denominator.

7.20.2 Member Function Documentation

7.20.2.1 getAcceptConflict()

```
bool saftlib::Condition_Proxy::getAcceptConflict ( )
```

Should conflicting actions be executed? Defaults to false.

Returns

Should conflicting actions be executed? Defaults to false

7.20.2.2 getAcceptDelayed()

```
bool saftlib::Condition_Proxy::getAcceptDelayed ( )
```

Should delayed actions be executed? Defaults to true.

Returns

Should delayed actions be executed? Defaults to true

7.20.2.3 getAcceptEarly()

```
bool saftlib::Condition_Proxy::getAcceptEarly ( )
```

Should early actions be executed? Defaults to false.

Returns

Should early actions be executed? Defaults to false

7.20.2.4 getAcceptLate()

```
bool saftlib::Condition_Proxy::getAcceptLate ( )
```

Should late actions be executed? Defaults to false -->

Returns

Should late actions be executed? Defaults to false -->

7.20.2.5 getActive()

```
bool saftlib::Condition_Proxy::getActive ( )
```

The condition should be actively matched against events.

Returns

The condition should be actively matched against events.

An inactive condition is not used to match against events. You can toggle the active state of this condition via this property, or if multiple Conditions should be atomically adjusted, use the ToggleActive method on the ActionSink.

7.20.2.6 getID()

```
uint64_t saftlib::Condition_Proxy::getID ( )
```

The event identifier which this condition matches against.

Returns

The event identifier which this condition matches against.

An incoming event matches if its ID and this ID agree on all the bits set in this condition's Mask.

7.20.2.7 getMask()

```
uint64_t saftlib::Condition_Proxy::getMask ( )
```

The mask used when comparing event IDs.

Returns

The mask used when comparing event IDs.

An incoming event matches if its ID and the ID property of this Condition agree on all the bits set in this Mask.

7.20.2.8 getOffset()

```
int64_t saftlib::Condition_Proxy::getOffset ( )
```

Added to an event's time to calculate the action's time.

Returns

Added to an event's time to calculate the action's time.

7.20.2.9 signal_dispatch()

```
bool saftlib::Condition_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

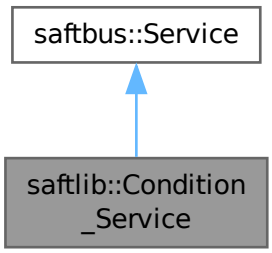
Reimplemented in saftlib::EmbeddedCPUCondition_Proxy, saftlib::OutputCondition_Proxy, saftlib::SCUbusCondition_Proxy, saftlib::SoftwareCondition_Proxy, and saftlib::WbmCondition_Proxy.

The documentation for this class was generated from the following files:

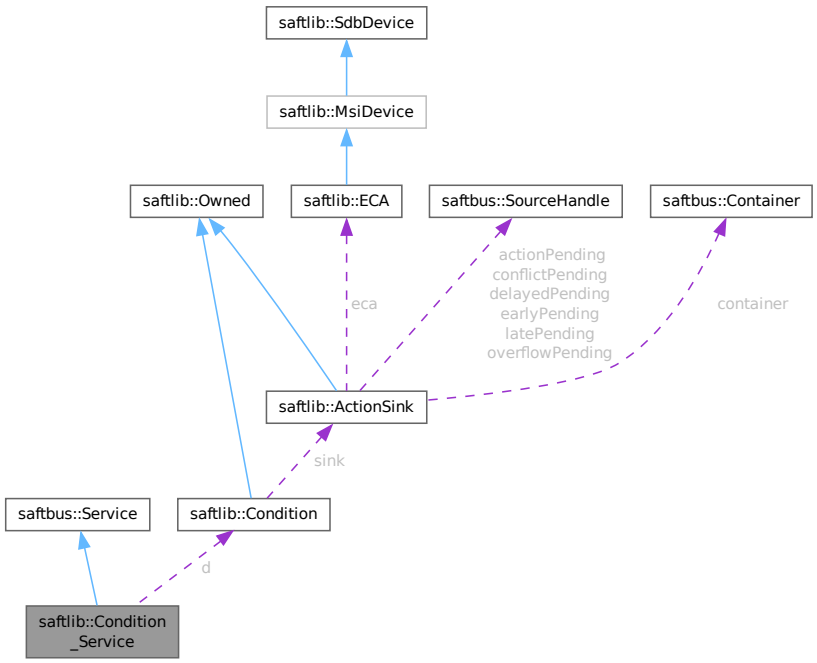
- src/Condition_Proxy.hpp
- src/Condition_Proxy.cpp

7.21 saftlib::Condition_Service Class Reference

Inheritance diagram for saftlib::Condition_Service:



Collaboration diagram for saftlib::Condition_Service:



Public Types

- typedef Condition **DriverType**

Public Member Functions

- **Condition_Service** (Condition *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- Condition * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.21.1 Member Function Documentation

7.21.1.1 call()

```
void saftlib::Condition_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/Condition_Service.hpp
- src/Condition_Service.cpp

7.22 ConditionContainer Struct Reference

Public Member Functions

- **ConditionContainer** (std::shared_ptr< saftlib::SoftwareCondition_Proxy > con)
- void **on_action** (uint64_t event, uint64_t param, saftlib::Time deadline, saftlib::Time executed, uint16_t flags)

Public Attributes

- std::shared_ptr< saftlib::SoftwareCondition_Proxy > **condition**

The documentation for this struct was generated from the following file:

- src/saft-testbench.cpp

7.23 saftbus::Container Class Reference

A Container of Service objects.

`#include <service.hpp>`

Classes

- struct Impl

Public Member Functions

- Container (ServerConnection *connection)

  create a Container for saftbus::Service objects
- void **add_additional_info_callback** (const std::string &name, std::function< std::string(void)> callback)

  add callback that returns a string which is shown up as additional information in get_status()
- void **remove_additional_info_callback** (const std::string &name)

  remove info callback. Plugins should clean up their callbacks when being unloaded
- unsigned create_object (const std::string &object_path, std::unique_ptr< Service > service)

  Insert a Service object and return the saftbus_object_id for this object.
- Service * **get_object** (const std::string &object_path)
- void destroy_service (Service *service)

  this function can be used by a Service to destroy itself
- bool call_service (unsigned saftbus_object_id, int client_fd, Deserializer &received, Serializer &send)

  call a Service identified by the saftbus_object_id
- void **remove_signal_fd** (int fd)
- void client_hung_up (int fd)

  iterate all owned services and remove the ones previously owned by client with this fd
- int get_calling_client_id () const

  some functions for ownership management. They can only be called whenever a client request is handled.
- void clear ()

  erase all objects in a safe manner
- Service * removal_helper (const std::string &object_path)

  checks if object_path exists and if the object itself or any of its children are owned
- int **register_proxy** (const std::string &object_path, const std::vector< std::string > interface_names, std::map< std::string, int > &interface_name2no_map, int client_fd, int signal_group_fd)
- void **unregister_proxy** (unsigned saftbus_object_id, int client_fd, int signal_group_fd)
- bool **load_plugin** (const std::string &so_filename, const std::vector< std::string > &args=std::vector< std::string >())
- bool **unload_plugin** (const std::string &so_filename, const std::vector< std::string > &args=std::vector< std::string >())
- bool remove_object (const std::string &object_path)

  remove an object
- void **quit** ()
- SaftbusInfo **get_status** ()

Friends

- class **Container_Service**

7.23.1 Detailed Description

A Container of Service objects.

Classes derived from Service can be stored here. One instance of Container is hold by the Connection object and all Service objects are available for remote Proxy objects to register and execute function calls through the Connection.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 Container()

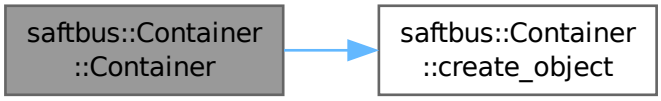
```
saftbus::Container::Container (
            ServerConnection * connection )
```

create a Container for saftbus::Service objects

Parameters

connection	The Connection object that owns the Container

Here is the call graph for this function:



7.23.3 Member Function Documentation

7.23.3.1 call_service()

```
bool saftbus::Container::call_service (
            unsigned saftbus_object_id,
            int client_fd,
            Deserializer & received,
            Serializer & send )
```

call a Service identified by the saftbus_object_id

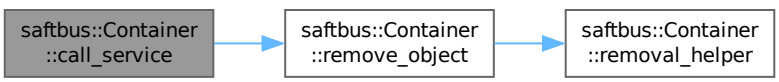
Parameters

saftbus_object_id	identifies the service object
client_fd	the file descriptor to the calling client
received	data that came from the client and is deserialized into function arguments
send	serialized return values that will be sent back to the client

Returns

false if the saftbus_object_id is unknown

Here is the call graph for this function:



7.23.3.2 clear()

```
void saftbus::Container::clear ( )
```

erase all objects in a safe manner

Children are erased before parents and younger objects are erased before older ones. Children and parents are identified by the object path: e.g. /grandparent/parent/child

7.23.3.3 client_hung_up()

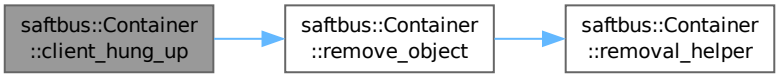
```
void saftbus::Container::client_hung_up (
            int fd )
```

iterate all owned services and remove the ones previously owned by client with this fd

Parameters

fd	the file descriptor that signaled a hung-up condition

Here is the call graph for this function:



7.23.3.4 create_object()

```
unsigned saftbus::Container::create_object (
            const std::string & object_path,
            std::unique_ptr< Service > service )
```

Insert a Service object and return the saftbus_object_id for this object.

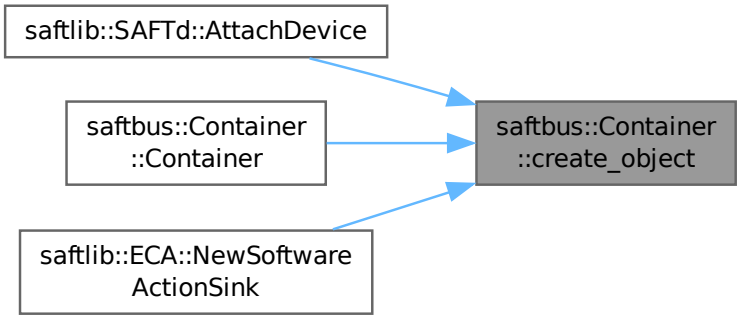
Parameters

object_path	the object path under which the Service object is available to Proxy objects.
service	A Service object

Returns

0 in case the object_path is already used by another Service object. The object_id if the Service object was successfully inserted into the Container

Here is the caller graph for this function:

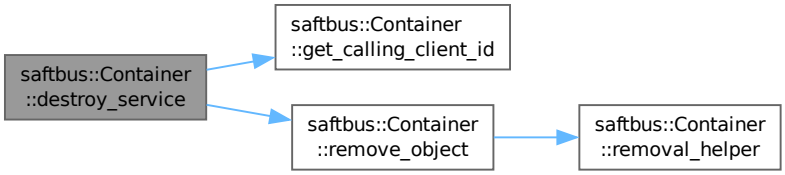


7.23.3.5 destroy_service()

```
void saftbus::Container::destroy_service (
            Service * service )
```

this function can be used by a Service to destroy itself

If this function is called during a remote function execution the destruction is delayed until the remote function call is completed. This is necessary because the Service is needed to complete the remote function call. Here is the call graph for this function:



7.23.3.6 get_calling_client_id()

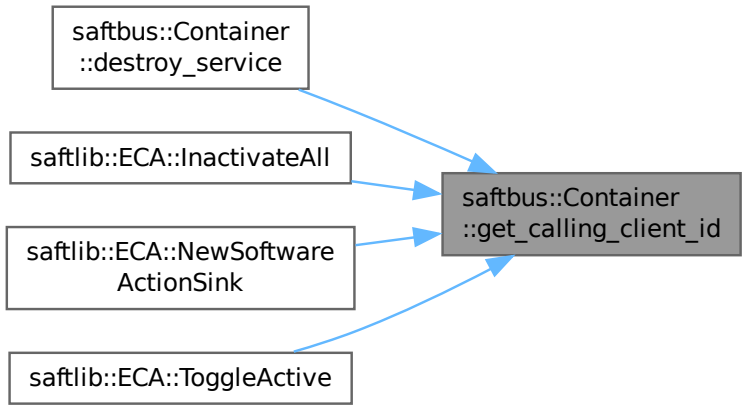
```
int saftbus::Container::get_calling_client_id ( ) const
```

some functions for ownership management. They can only be called whenever a client request is handled.

Returns

-1 if no client is calling, the client id if a client call is in progress

Here is the caller graph for this function:



7.23.3.7 removal_helper()

```
Service * saftbus::Container::removal_helper (
            const std::string & object_path )
```

checks if object_path exists and if the object itself or any of its children are owned

If the object does not exist or it or any of its children are owned, an exception is thrown.

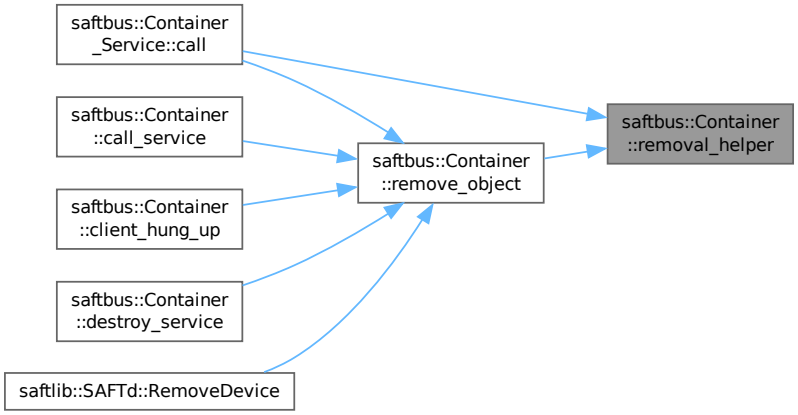
Parameters

object_path	the object path of the service to be removed

Returns

A pointer to the service object under object_path.

Here is the caller graph for this function:



7.23.3.8 remove_object()

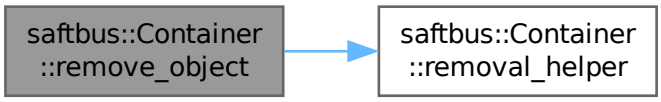
```
bool saftbus::Container::remove_object (
            const std::string & object_path )
```

remove an object

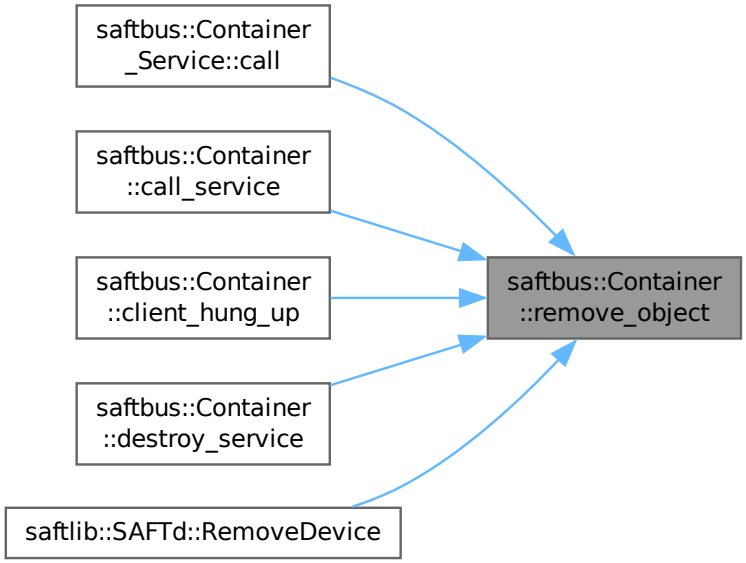
Parameters

object_path	the object path of the service object to be removed

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

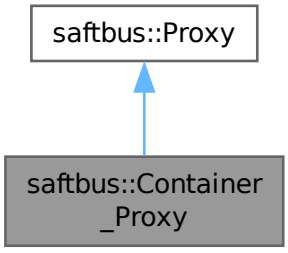
- saftbus/service.hpp
- saftbus/service.cpp

7.24 saftbus::Container_Proxy Class Reference

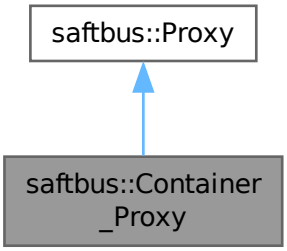
A Container of Service objects.

`#include <client.hpp>`

Inheritance diagram for saftbus::Container_Proxy:



Collaboration diagram for saftbus::Container_Proxy:



Public Member Functions

- **Container_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group, const std::vector< std::string > &interface_names=std::vector< std::string >())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- bool **load_plugin** (const std::string &so_filename, const std::vector< std::string > &plugin_args=std::vector< std::string >())
- bool **unload_plugin** (const std::string &so_filename, const std::vector< std::string > &plugin_args=std::vector< std::string >())
- bool **remove_object** (const std::string &object_path)
- void **quit** ()
- SaftbusInfo **get_status** ()

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< Container_Proxy > **create** (const std::string &object_path="/saftbus", saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=std::vector< std::string >())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.24.1 Detailed Description

A Container of Service objects.

Classes derived from Service can be stored here. One instance of Container is hold by the Connection object and all Service objects are available for remote Proxy objects to register and execute function calls through the Connection.

This was created by saftbus-gen from the Container class and manually copied to this place. The Container class is special and the Proxy code is slightly different from all other Proxy classes.

7.24.2 Member Function Documentation

7.24.2.1 signal_dispatch()

```
bool saftbus::Container_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

The documentation for this class was generated from the following files:

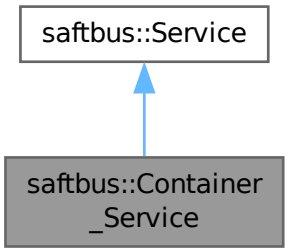
- saftbus/client.hpp
- saftbus/client.cpp

7.25 saftbus::Container_Service Class Reference

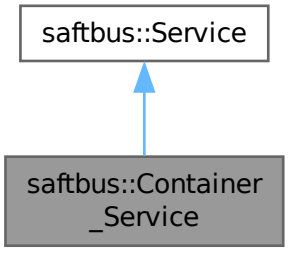
created by saftbus-gen from class Container and copied here

```
#include <service.hpp>
```

Inheritance diagram for saftbus::Container_Service:



Collaboration diagram for saftbus::Container_Service:



Public Types

- typedef Container **DriverType**

Public Member Functions

- **Container_Service** (Container *instance, std::function< void()> destruction_callback=nullptr)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.25.1 Detailed Description

created by saftbus-gen from class Container and copied here

7.25.2 Member Function Documentation

7.25.2.1 call()

```
void saftbus::Container_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

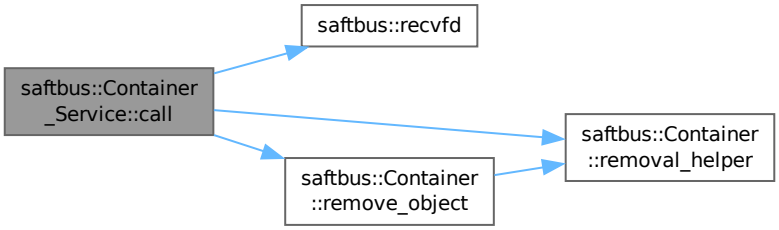
Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

- saftbus/service.hpp
- saftbus/service.cpp

7.26 saftbus::Deserializer Class Reference

Simple deserializer.

`#include <saftbus.hpp>`

Public Member Functions

- **Deserializer** (int reserve=4096)
- bool **read_from** (int fd)
- template<typename T >
  std::enable_if< std::is_base_of< SerDesAble, T >::value >::type **get** (T &val) const
- template<typename T >
  std::enable_if<!std::is_base_of< SerDesAble, T >::value >::type **get** (T &val) const
- template<typename T >
  void **get** (std::vector< T > &std_vector) const
- template<typename T >
  void **get** (std::vector< std::vector< T, std::allocator< T > >, std::allocator< std::vector< T, std::allocator< T > > > > &std_vector_vector) const
- void **get** (std::string &std_string) const
- void **get** (std::vector< std::string > &vector_string) const
- template<typename K , typename V >
  void **get** (std::map< K, V > &std_map) const
- void **save** () const
- void **restore** () const

7.26.1 Detailed Description

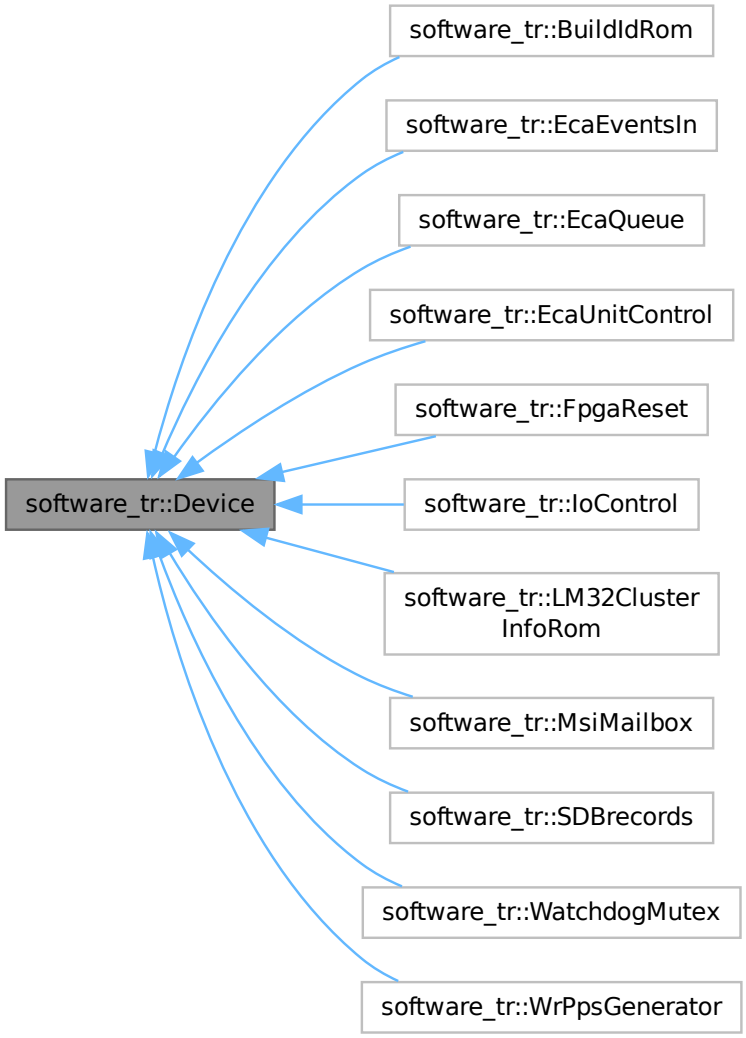
Simple deserializer.

The documentation for this class was generated from the following files:

- saftbus/saftbus.hpp
- saftbus/saftbus.cpp

7.27 software_tr::Device Class Reference

Inheritance diagram for software_tr::Device:



Public Member Functions

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

The documentation for this class was generated from the following file:

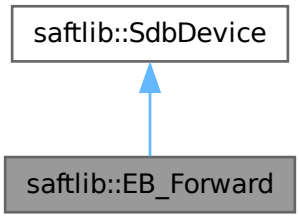
- src/saft-software-tr.cpp

7.28 saftlib::EB_Forward Class Reference

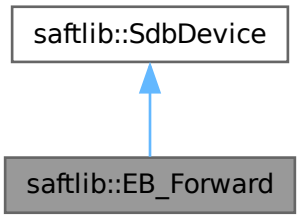
Maintains a pseudo-terminal device that mimics a serial etherbone device.

`#include <eb-forward.hpp>`

Inheritance diagram for saftlib::EB_Forward:



Collaboration diagram for saftlib::EB_Forward:



Public Member Functions

- **EB_Forward** (const std::string &eb_name, etherbone::Device &device)
- bool **accept_connection** (int condition)
- std::string eb_forward_path ()

  return the name of the pseudo-terminal device

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.28.1 Detailed Description

Maintains a pseudo-terminal device that mimics a serial etherbone device.

All data from the created pseudo terminal (e.g. /dev/pts/14) is read, split-up into complete etherbone packages and redirected to the real hardware. When the real hardware responds to the etherbone request, the response is written back to the pseudo-terminal device. This effectively allows using eb-tools, such as eb-ls on serial devices, even when the device is occupied the TimingReceiver object.

7.28.2 Member Function Documentation

7.28.2.1 eb_forward_path()

```
std::string saftlib::EB_Forward::eb_forward_path ( )
```

return the name of the pseudo-terminal device

Returns

the name of the created pseudo-terminal

The documentation for this class was generated from the following files:

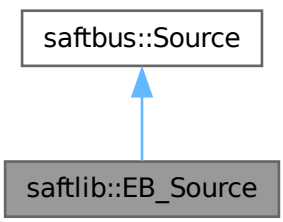
- src/eb-forward.hpp
- src/eb-forward.cpp

7.29 saftlib::EB_Source Class Reference

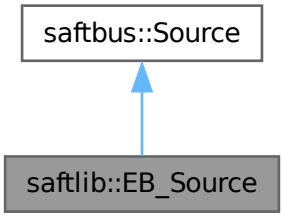
an etherbone event source for the saftbus::Loop

```
#include <eb-source.hpp>
```

Inheritance diagram for saftlib::EB_Source:



Collaboration diagram for saftlib::EB_Source:



Public Member Functions

- **EB_Source** (etherbone::Socket socket_)
- bool prepare (std::chrono::milliseconds &timeout_ms)
- bool check ()
- bool dispatch ()
- std::string type ()

Public Member Functions inherited from saftbus::Source

- virtual bool **prepare** (std::chrono::milliseconds &timeout_ms)=0
- virtual bool **check** ()=0
- virtual bool **dispatch** ()=0
- virtual std::string **type** ()=0
- long **get_id** ()

Static Public Member Functions

- static int **add_fd** (eb_user_data_t, eb_descriptor_t, uint8_t mode)
- static int **get_fd** (eb_user_data_t, eb_descriptor_t, uint8_t mode)

Additional Inherited Members

Protected Member Functions inherited from saftbus::Source

- void **add_poll** (pollfd *pfd)
- void **remove_poll** (pollfd *pfd)
- void **clear_poll** ()

7.29.1 Detailed Description

an etherbone event source for the saftbus::Loop

7.29.2 Member Function Documentation

7.29.2.1 check()

```
bool saftlib::EB_Source::check ( )  [virtual]
```

Implements saftbus::Source.

7.29.2.2 dispatch()

```
bool saftlib::EB_Source::dispatch ( )  [virtual]
```

Implements saftbus::Source.

7.29.2.3 prepare()

```
bool saftlib::EB_Source::prepare (
            std::chrono::milliseconds & timeout_ms )  [virtual]
```

Implements saftbus::Source.

7.29.2.4 type()

```
std::string saftlib::EB_Source::type ( )  [virtual]
```

Implements saftbus::Source.

The documentation for this class was generated from the following files:

- src/eb-source.hpp
- src/eb-source.cpp

7.30 software_tr::EBslave Class Reference

Public Member Functions

- void **init** ()
- void **shutdown** ()
- **EBslave** (bool stop_until_connected, uint32_t sdb_adr, uint32_t msi_addr_first, uint32_t msi_addr_last)
- std::string **pts_name** ()
- void **fill_input_buffer** ()
- bool **next_word** (uint32_t &result)
- int **master_out** (std_logic_t *cyc, std_logic_t *stb, std_logic_t *we, int *adr, int *dat, int *sel)
- int **handle_pass_through** ()
- void **send_output_buffer** ()
- int **master_in** (std_logic_t ack, std_logic_t err, std_logic_t rty, std_logic_t stall, int dat)
- void **msi_slave_out** (std_logic_t *ack, std_logic_t *err, std_logic_t *rty, std_logic_t *stall, int *dat)
- void **msi_slave_in** (std_logic_t cyc, std_logic_t stb, std_logic_t we, int adr, int dat, int sel)
- void **push_msi** (uint32_t adr, uint32_t dat)

The documentation for this class was generated from the following file:

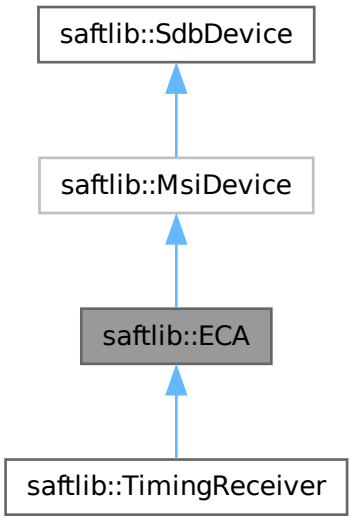
- src/saft-software-tr.cpp

7.31 saftlib::ECA Class Reference

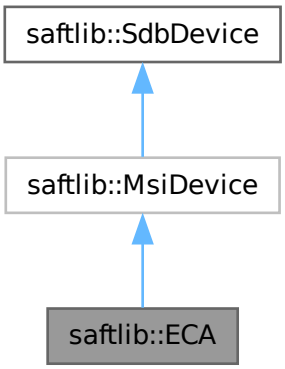
ECA (Event Condition Action) ECA is a hardwar unit cabable of executing actions at a given time (1 ns resolution) in response to events that meet a condition. An event contains a 64-bit timestamp, 64-bit id, 64-bit flags A condition contains a 64-bit id, 64-bit prefix-mask, 64-bit time offset Action can be one of the following:

`#include <ECA.hpp>`

Inheritance diagram for saftlib::ECA:



Collaboration diagram for saftlib::ECA:



Public Member Functions

- const std::string & **get_object_path** ()
- etherbone::Device & **get_device** ()
- void **compile** ()
- **ECA** (SAFTd &saftd, etherbone::Device &device, const std::string &object_path, saftbus::Container *container)
- bool **addActionSink** (int channel, std::unique_ptr< ActionSink > sink)

  add sink and let ECA take ownership of the sink object
- uint16_t **updateMostFull** (unsigned channel)
- void **removeSowftwareActionSink** (SoftwareActionSink *sas)
- uint64_t ReadRawCurrentTime () const

  The current time of the timingreceiver.
- std::string NewSoftwareActionSink (const std::string &name)

  Create a new SoftwareActionSink.
- std::map< std::string, std::string > getSoftwareActionSinks () const

  A list of all current SoftwareActionSinks.
- std::map< std::string, std::string > getSCUbusActionSinks () const

  A list of all SCUbusActionSinks.
- std::map< std::string, std::string > getEmbeddedCPUActionSinks () const

  A list of all EmbeddedCPUActionSinks.
- std::map< std::string, std::string > getWbmActionSinks () const

  A list of all WbmActionSinks.
- SoftwareActionSink * getSoftwareActionSink (const std::string &sas_obj_path)

  get a pointer to a SoftwareActionSink in a stand-alone application
- void **ToggleActive** ()

  activate all inactive owned conditions and inactivate all active owned conditions
- void **InactivateAll** ()

  deactivate all owned conditions
- std::map< std::string, std::string > getOutputs () const

  A list of all the high/low outputs on the receiver.
- uint32_t getFree () const

  The number of additional conditions that may be activated.
- void **resetMostFull** (unsigned channel)
- Output * **getOutput** (const std::string &output_obj_path)

Public Member Functions inherited from saftlib::MsiDevice

- **MsiDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID)

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_found=true)

Friends

- class **ActionSink**

Additional Inherited Members

Protected Attributes inherited from saftlib::MsiDevice

- etherbone::sdb_msi_device **msi_device**

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.31.1 Detailed Description

ECA (Event Condition Action) ECA is a hardwar unit cabable of executing actions at a given time (1 ns resolution) in response to events that meet a condition. An event contains a 64-bit timestamp, 64-bit id, 64-bit flags A condition contains a 64-bit id, 64-bit prefix-mask, 64-bit time offset Action can be one of the following:

- Output: change output driver (rising or falling edge on one of the GPIOs)
- SoftwareActionSink: send the content of the event to host system (store the event content in some registers and send an MSI so that the host can read the event information)
- execute a preconfigured wishbone access
- SCUbusActionSink: write a tag on SCU bus
- ECPU : send content of the event to embedded CPU

Hardware supports are up to 32 SoftwareActionSink at the same time. The hardware has one single Output channel configured for the host system with 32 sub-channels for the different SoftwareActionSinks. Each SoftwareAction↩ Sinks occupies one of the sub-channels. This implementation uses an std::vector< std::unique_ptr< ActionSink> > which is resized in the constructor to the number of available sub-channels. Each invalid unique_ptr<Action↩ Sink> indicates an un-occupied sub-channel. Each SoftwareActionSink can have many conditions. On execution, ECA writes a tag value into the ActionSink that refers to the condition that matched the incoming event.

7.31.2 Member Function Documentation

7.31.2.1 getEmbeddedCPUActionSinks()

```
std::map< std::string, std::string > saftlib::ECA::getEmbeddedCPUActionSinks ( ) const
```

A list of all EmbeddedCPUActionSinks.

Returns

A list of all EmbeddedCPUActionSinks

7.31.2.2 getFree()

```
uint32_t saftlib::ECA::getFree ( ) const
```

The number of additional conditions that may be activated.

The ECA has limited hardware resources in its match table.

Returns

number of additional conditions that may be activated.

7.31.2.3 getOutputs()

```
std::map< std::string, std::string > saftlib::ECA::getOutputs ( ) const
```

A list of all the high/low outputs on the receiver.

Returns

A list of all the high/low outputs on the receiver.

Each path refers to an object of type Output.

7.31.2.4 getSCUbusActionSinks()

```
std::map< std::string, std::string > saftlib::ECA::getSCUbusActionSinks ( ) const
```

A list of all SCUbusActionSinks.

Returns

A list of all SCUbusActionSinks

7.31.2.5 getSoftwareActionSink()

```
SoftwareActionSink * saftlib::ECA::getSoftwareActionSink (
            const std::string & sas_obj_path )
```

get a pointer to a SoftwareActionSink in a stand-alone application

Parameters

sas_obj_path	Object path of the SoftwareActionSink

Returns

pointer to a SoftwareActionSink

7.31.2.6 getSoftwareActionSinks()

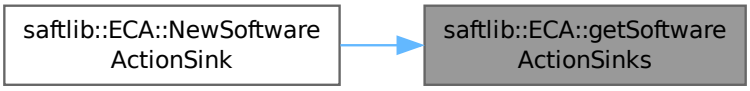
```
std::map< std::string, std::string > saftlib::ECA::getSoftwareActionSinks ( ) const
```

A list of all current SoftwareActionSinks.

Returns

A list of all current SoftwareActionSinks.

Typically, these SoftwareActionSinks will be owned by their processes and not of much interest to others. Therefore, many of the entries here may be of no interest to a particular client. However, it is possible for a SoftwareActionSink to be Disowned, in which case it may be persistent and shared between programs under a well known name. Here is the caller graph for this function:



7.31.2.7 getWbmActionSinks()

```
std::map< std::string, std::string > saftlib::ECA::getWbmActionSinks ( ) const
```

A list of all WbmActionSinks.

Returns

A list of all WbmActionSinks

7.31.2.8 NewSoftwareActionSink()

```
std::string saftlib::ECA::NewSoftwareActionSink (
            const std::string & name )
```

Create a new SoftwareActionSink.

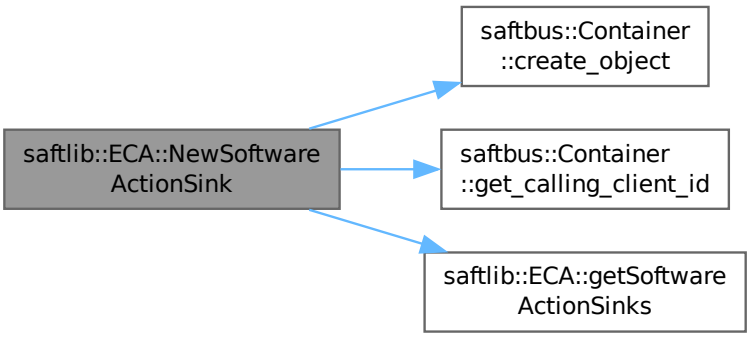
Parameters

name	A name for the SoftwareActionSink. Can be left blank.

Returns

Object path to the created SoftwareActionSink.

SoftwareActionSinks allow a program to create conditions that match incoming timing events. These conditions may have callback methods attached to them in order to receive notification. The returned path corresponds to a SoftwareActionSink that is owned by the process which claimed it, and can thus be certain that no other processes can interfere with the results. Here is the call graph for this function:



7.31.2.9 ReadRawCurrentTime()

```
uint64_t saftlib::ECA::ReadRawCurrentTime ( ) const
```

The current time of the timingreceiver.

Returns

the current time of the timingreceiver

The result is the WhiteRabbit timesamps in nanoseconds. It is not checked if WhiteRabbit core is locked or not.

The documentation for this class was generated from the following files:

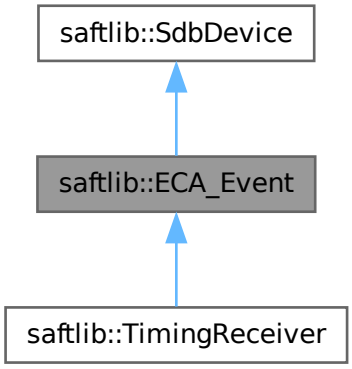
- src/ECA.hpp
- src/ECA.cpp

7.32 saftlib::ECA_Event Class Reference

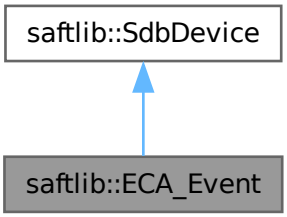
ECA_Event (Event Injection interface of ECA) ECA_Event provides a method to inject events into ECA.

`#include <ECA_Event.hpp>`

Inheritance diagram for saftlib::ECA_Event:



Collaboration diagram for saftlib::ECA_Event:



Public Member Functions

- **ECA_Event** (etherbone::Device &device, saftbus::Container *container=nullptr)
- void InjectEventRaw (uint64_t event, uint64_t param, uint64_t time) const

  Simulate the receipt of a timing event.

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.32.1 Detailed Description

ECA_Event (Event Injection interface of ECA) ECA_Event provides a method to inject events into ECA.

7.32.2 Member Function Documentation

7.32.2.1 InjectEventRaw()

```
void saftlib::ECA_Event::InjectEventRaw (
            uint64_t event,
            uint64_t param,
            uint64_t time ) const
```

Simulate the receipt of a timing event.

Parameters

event	The event identifier which is matched against Conditions
param	The parameter field, whose meaning depends on the event ID.
time	The execution time for the event, added to condition offsets.

Sometimes it is useful to simulate the receipt of a timing event. This allows software to test that configured conditions lead to the desired behaviour without needing the data master to send anything.

The documentation for this class was generated from the following files:

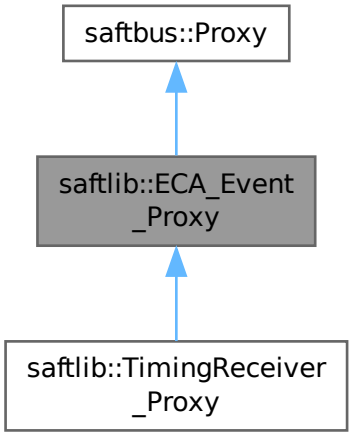
- src/ECA_Event.hpp
- src/ECA_Event.cpp

7.33 saftlib::ECA_Event_Proxy Class Reference

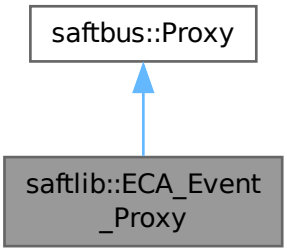
ECA_Event (Event Injection interface of ECA) ECA_Event provides a method to inject events into ECA.

```
#include <ECA_Event_Proxy.hpp>
```

Inheritance diagram for saftlib::ECA_Event_Proxy:



Collaboration diagram for saftlib::ECA_Event_Proxy:



Public Member Functions

- **ECA_Event_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void InjectEventRaw (uint64_t event, uint64_t param, uint64_t time)

  Simulate the receipt of a timing event.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< ECA_Event_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.33.1 Detailed Description

ECA_Event (Event Injection interface of ECA) ECA_Event provides a method to inject events into ECA.

7.33.2 Member Function Documentation

7.33.2.1 InjectEventRaw()

```
void saftlib::ECA_Event_Proxy::InjectEventRaw (
            uint64_t event,
            uint64_t param,
            uint64_t time )
```

Simulate the receipt of a timing event.

Parameters

event	The event identifier which is matched against Conditions
param	The parameter field, whose meaning depends on the event ID.
time	The execution time for the event, added to condition offsets.

Sometimes it is useful to simulate the receipt of a timing event. This allows software to test that configured conditions lead to the desired behaviour without needing the data master to send anything.

7.33.2.2 signal_dispatch()

```
bool saftlib::ECA_Event_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

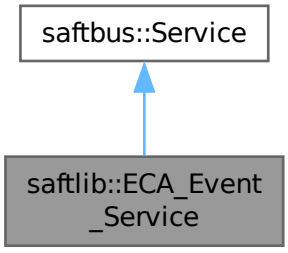
Reimplemented in saftlib::TimingReceiver_Proxy.

The documentation for this class was generated from the following files:

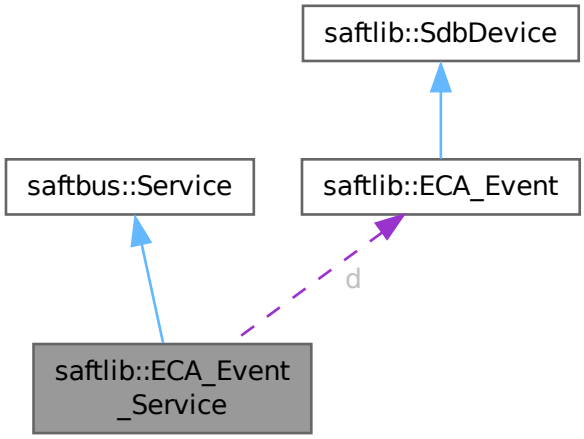
- src/ECA_Event_Proxy.hpp
- src/ECA_Event_Proxy.cpp

7.34 saftlib::ECA_Event_Service Class Reference

Inheritance diagram for saftlib::ECA_Event_Service:



Collaboration diagram for saftlib::ECA_Event_Service:



Public Types

- typedef ECA_Event **DriverType**

Public Member Functions

- **ECA_Event_Service** (ECA_Event *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- ECA_Event * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.34.1 Member Function Documentation

7.34.1.1 call()

```
void saftlib::ECA_Event_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/ECA_Event_Service.hpp
- src/ECA_Event_Service.cpp

7.35 saftlib::ECA_OpenClose Struct Reference

Public Attributes

- uint64_t **key**
- bool **open**
- uint64_t **subkey**
- int64_t **offset**
- uint32_t **tag**
- uint8_t **flags**
- unsigned **channel**
- unsigned **num**

The documentation for this struct was generated from the following file:

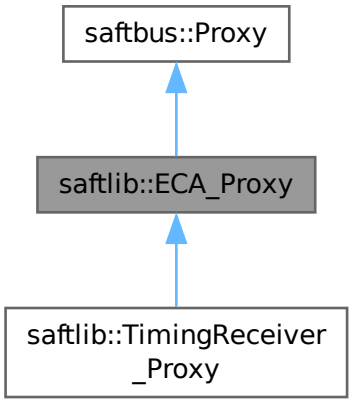
- src/ECA.cpp

7.36 saftlib::ECA_Proxy Class Reference

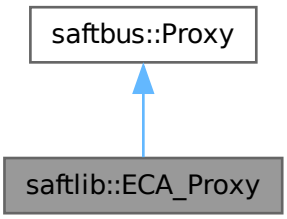
ECA (Event Condition Action) ECA is a hardwar unit cabable of executing actions at a given time (1 ns resolution) in response to events that meet a condition. An event contains a 64-bit timestamp, 64-bit id, 64-bit flags A condition contains a 64-bit id, 64-bit prefix-mask, 64-bit time offset Action can be one of the following:

```
#include <ECA_Proxy.hpp>
```

Inheritance diagram for saftlib::ECA_Proxy:



Collaboration diagram for saftlib::ECA_Proxy:



Public Member Functions

- **ECA_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t ReadRawCurrentTime ()

  The current time of the timingreceiver.
- std::string NewSoftwareActionSink (const std::string &name)

  Create a new SoftwareActionSink.
- std::map< std::string, std::string > getSoftwareActionSinks ()

  A list of all current SoftwareActionSinks.
- std::map< std::string, std::string > getSCUbusActionSinks ()

  A list of all SCUbusActionSinks.
- std::map< std::string, std::string > getEmbeddedCPUActionSinks ()

  A list of all EmbeddedCPUActionSinks.
- std::map< std::string, std::string > getWbmActionSinks ()

  A list of all WbmActionSinks.
- void **ToggleActive** ()

  activate all inactive owned conditions and inactivate all active owned conditions
- void **InactivateAll** ()

  deactivate all owned conditions
- std::map< std::string, std::string > getOutputs ()

  A list of all the high/low outputs on the receiver.
- uint32_t getFree ()

  The number of additional conditions that may be activated.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< ECA_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.36.1 Detailed Description

ECA (Event Condition Action) ECA is a hardwar unit cabable of executing actions at a given time (1 ns resolution) in response to events that meet a condition. An event contains a 64-bit timestamp, 64-bit id, 64-bit flags A condition contains a 64-bit id, 64-bit prefix-mask, 64-bit time offset Action can be one of the following:

- Output: change output driver (rising or falling edge on one of the GPIOs)
- SoftwareActionSink: send the content of the event to host system (store the event content in some registers and send an MSI so that the host can read the event information)
- execute a preconfigured wishbone access
- SCUbusActionSink: write a tag on SCU bus
- ECPU : send content of the event to embedded CPU

Hardware supports are up to 32 SoftwareActionSink at the same time. The hardware has one single Output channel configured for the host system with 32 sub-channels for the different SoftwareActionSinks. Each SoftwareActionSinks occupies one of the sub-channels. This implementation uses an std::vector< std::unique_ptr< ActionSink> > which is resized in the constructor to the number of available sub-channels. Each invalid unique_ptr<ActionSink> indicates an un-occupied sub-channel. Each SoftwareActionSink can have many conditions. On execution, ECA writes a tag value into the ActionSink that refers to the condition that matched the incoming event.

7.36.2 Member Function Documentation

7.36.2.1 getEmbeddedCPUActionSinks()

```
std::map< std::string, std::string > saftlib::ECA_Proxy::getEmbeddedCPUActionSinks ( )
```

A list of all EmbeddedCPUActionSinks.

Returns

A list of all EmbeddedCPUActionSinks

7.36.2.2 getFree()

```
uint32_t saftlib::ECA_Proxy::getFree ( )
```

The number of additional conditions that may be activated.

The ECA has limited hardware resources in its match table.

Returns

number of additional conditions that may be activated.

7.36.2.3 getOutputs()

```
std::map< std::string, std::string > saftlib::ECA_Proxy::getOutputs ( )
```

A list of all the high/low outputs on the receiver.

Returns

A list of all the high/low outputs on the receiver.

Each path refers to an object of type Output.

7.36.2.4 getSCUbusActionSinks()

```
std::map< std::string, std::string > saftlib::ECA_Proxy::getSCUbusActionSinks ( )
```

A list of all SCUbusActionSinks.

Returns

A list of all SCUbusActionSinks

7.36.2.5 getSoftwareActionSinks()

```
std::map< std::string, std::string > saftlib::ECA_Proxy::getSoftwareActionSinks ( )
```

A list of all current SoftwareActionSinks.

Returns

A list of all current SoftwareActionSinks.

Typically, these SoftwareActionSinks will be owned by their processes and not of much interest to others. Therefore, many of the entries here may be of no interest to a particular client. However, it is possible for a SoftwareActionSink to be Disowned, in which case it may be persistent and shared between programs under a well known name.

7.36.2.6 getWbmActionSinks()

```
std::map< std::string, std::string > saftlib::ECA_Proxy::getWbmActionSinks ( )
```

A list of all WbmActionSinks.

Returns

A list of all WbmActionSinks

7.36.2.7 NewSoftwareActionSink()

```
std::string saftlib::ECA_Proxy::NewSoftwareActionSink (
            const std::string & name )
```

Create a new SoftwareActionSink.

Parameters

name	A name for the SoftwareActionSink. Can be left blank.

Returns

Object path to the created SoftwareActionSink.

SoftwareActionSinks allow a program to create conditions that match incoming timing events. These conditions may have callback methods attached to them in order to receive notification. The returned path corresponds to a SoftwareActionSink that is owned by the process which claimed it, and can thus be certain that no other processes can interfere with the results.

7.36.2.8 ReadRawCurrentTime()

```
uint64_t saftlib::ECA_Proxy::ReadRawCurrentTime ( )
```

The current time of the timingreceiver.

Returns

the current time of the timingreceiver

The result is the WhiteRabbit timesamps in nanoseconds. It is not checked if WhiteRabbit core is locked or not.

7.36.2.9 signal_dispatch()

```
bool saftlib::ECA_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

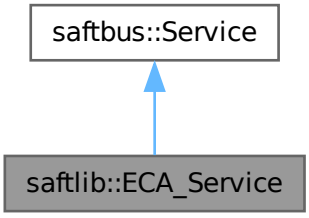
Reimplemented in saftlib::TimingReceiver_Proxy.

The documentation for this class was generated from the following files:

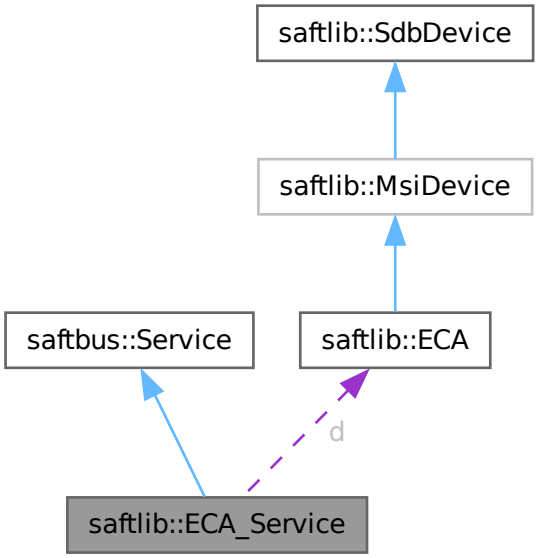
- src/ECA_Proxy.hpp
- src/ECA_Proxy.cpp

7.37 saftlib::ECA_Service Class Reference

Inheritance diagram for saftlib::ECA_Service:



Collaboration diagram for saftlib::ECA_Service:



Public Types

- typedef ECA **DriverType**

Public Member Functions

- **ECA_Service** (ECA *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- ECA * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.37.1 Member Function Documentation

7.37.1.1 call()

```
void saftlib::ECA_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

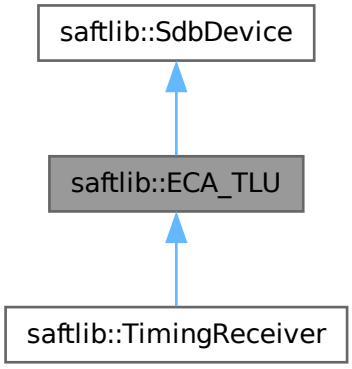
- src/ECA_Service.hpp
- src/ECA_Service.cpp

7.38 saftlib::ECA_TLU Class Reference

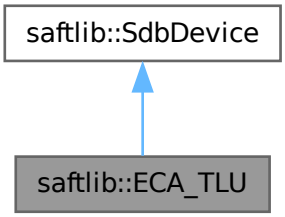
Interface to the ECA_TLU SdbInterface.

`#include <ECA_TLU.hpp>`

Inheritance diagram for saftlib::ECA_TLU:



Collaboration diagram for saftlib::ECA_TLU:



Public Member Functions

- **ECA_TLU** (etherbone::Device &device, saftbus::Container *container=nullptr)
- void **addInput** (std::unique_ptr< Input > input)

  add source and let ECA take ownership of the sink object
- std::map< std::string, std::string > **getInputs** () const
- void **configInput** (unsigned channel, bool enable, uint64_t event, uint32_t stable)

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.38.1 Detailed Description

Interface to the ECA_TLU SdbInterface.

The documentation for this class was generated from the following files:

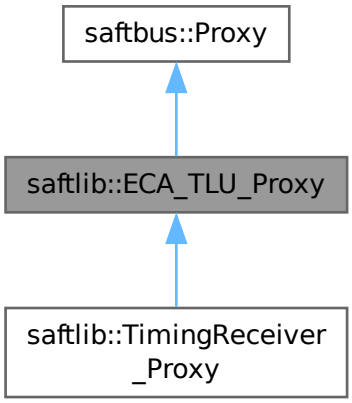
- src/ECA_TLU.hpp
- src/ECA_TLU.cpp

7.39 saftlib::ECA_TLU_Proxy Class Reference

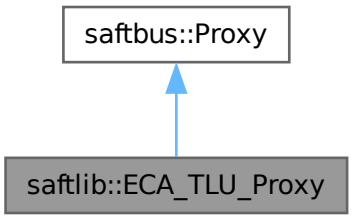
Interface to the ECA_TLU SdbInterface.

`#include <ECA_TLU_Proxy.hpp>`

Inheritance diagram for saftlib::ECA_TLU_Proxy:



Collaboration diagram for saftlib::ECA_TLU_Proxy:



Public Member Functions

- **ECA_TLU_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::map< std::string, std::string > **getInputs** ()

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< ECA_TLU_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.

- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.39.1 Detailed Description

Interface to the ECA_TLU SdbInterface.

7.39.2 Member Function Documentation

7.39.2.1 signal_dispatch()

```
bool saftlib::ECA_TLU_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

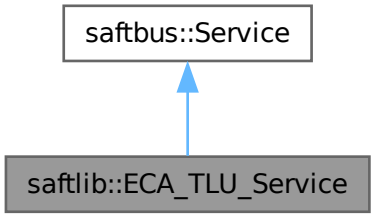
Reimplemented in saftlib::TimingReceiver_Proxy.

The documentation for this class was generated from the following files:

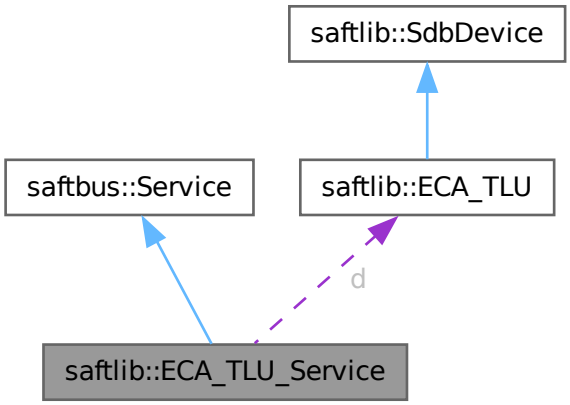
- src/ECA_TLU_Proxy.hpp
- src/ECA_TLU_Proxy.cpp

7.40 saftlib::ECA_TLU_Service Class Reference

Inheritance diagram for saftlib::ECA_TLU_Service:



Collaboration diagram for saftlib::ECA_TLU_Service:



Public Types

- typedef ECA_TLU **DriverType**

Public Member Functions

- **ECA_TLU_Service** (ECA_TLU *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- ECA_TLU * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.40.1 Member Function Documentation

7.40.1.1 call()

```
void saftlib::ECA_TLU_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

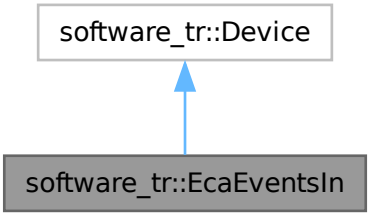
Implements saftbus::Service.

The documentation for this class was generated from the following files:

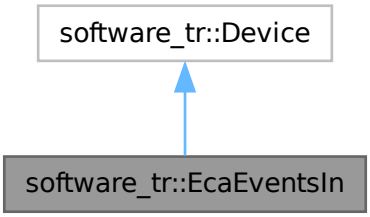
- src/ECA_TLU_Service.hpp
- src/ECA_TLU_Service.cpp

7.41 software_tr::EcaEventsIn Class Reference

Inheritance diagram for software_tr::EcaEventsIn:



Collaboration diagram for software_tr::EcaEventsIn:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0x8752bf45 }

Public Member Functions

- **EcaEventsIn** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool write_access (uint32_t adr, int sel, uint32_t dat)

Public Member Functions inherited from software_tr::Device

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.41.1 Member Function Documentation

7.41.1.1 contains()

```
bool software_tr::EcaEventsIn::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.41.1.2 write_access()

```
bool software_tr::EcaEventsIn::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

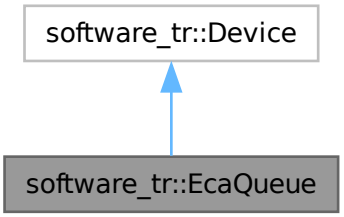
Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

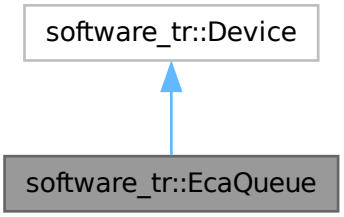
- src/saft-software-tr.cpp

7.42 software_tr::EcaQueue Class Reference

Inheritance diagram for software_tr::EcaQueue:



Collaboration diagram for software_tr::EcaQueue:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0xd5a3faea }

Public Member Functions

- **EcaQueue** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool read_access (uint32_t adr, int sel, uint32_t *dat_out)
- bool write_access (uint32_t adr, int sel, uint32_t dat)

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.42.1 Member Function Documentation

7.42.1.1 contains()

```
bool software_tr::EcaQueue::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.42.1.2 read_access()

```
bool software_tr::EcaQueue::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

7.42.1.3 write_access()

```
bool software_tr::EcaQueue::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

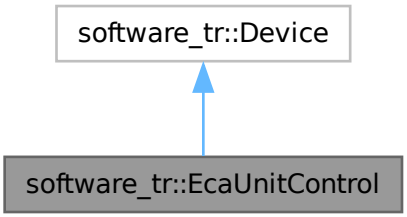
Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

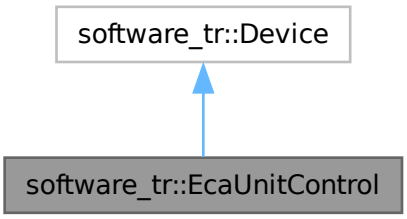
- src/saft-software-tr.cpp

7.43 software_tr::EcaUnitControl Class Reference

Inheritance diagram for software_tr::EcaUnitControl:



Collaboration diagram for software_tr::EcaUnitControl:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0xb2afc251 }

Public Member Functions

- **EcaUnitControl** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool read_access (uint32_t adr, int sel, uint32_t ∗dat_out)
- bool **is_prefix_mask** (uint64_t mask)
- uint64_t **fix_mask** (uint64_t mask)
- bool write_access (uint32_t adr, int sel, uint32_t dat)

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t ∗dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.43.1 Member Function Documentation

7.43.1.1 contains()

```
bool software_tr::EcaUnitControl::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.43.1.2 read_access()

```
bool software_tr::EcaUnitControl::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

7.43.1.3 write_access()

```
bool software_tr::EcaUnitControl::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

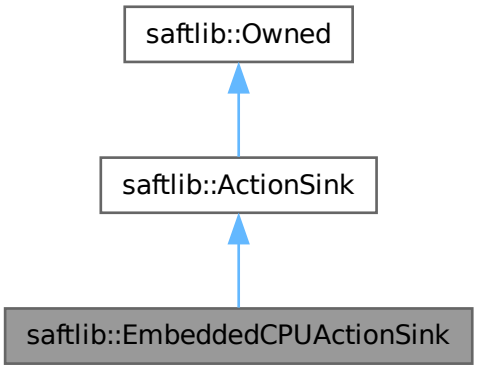
- src/saft-software-tr.cpp

7.44 saftlib::EmbeddedCPUActionSink Class Reference

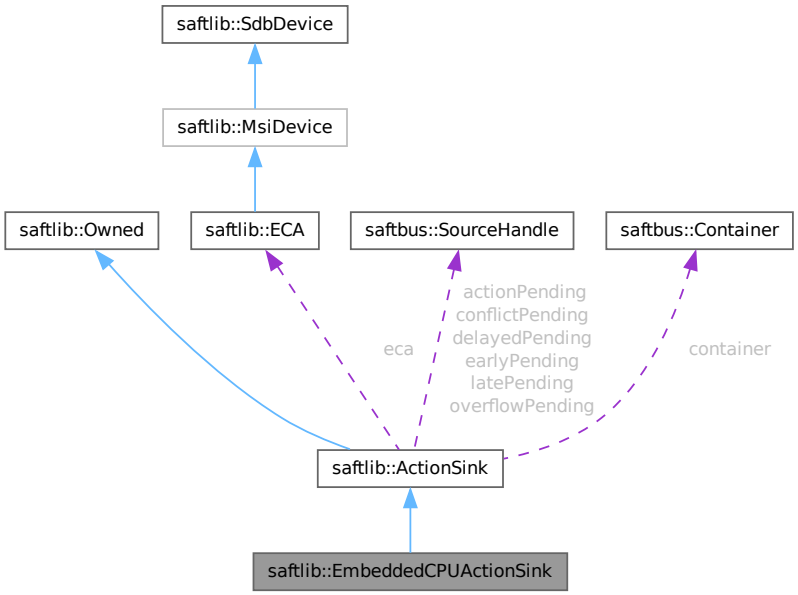
An output through which EmbeddedCPU actions flow.

`#include <EmbeddedCPUActionSink.hpp>`

Inheritance diagram for saftlib::EmbeddedCPUActionSink:



Collaboration diagram for saftlib::EmbeddedCPUActionSink:



Public Member Functions

- **EmbeddedCPUActionSink** (ECA &eca, const std::string &object_path, const std::string &name, unsigned channel, saftbus::Container *container=nullptr)
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag)

  Create a condition to match incoming events.

Public Member Functions inherited from saftlib::ActionSink

- ActionSink (ECA &eca, const std::string &action_sink_object_path, const std::string &name, unsigned channel, unsigned num, saftbus::Container *container=nullptr)

  ActionSink constructor.
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions () const

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions () const

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions () const

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset () const

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset () const

Maximum allowed offset (nanoseconds) usable in NewCondition.

- void **setMaxOffset** (int64_t val)
- uint64_t getLatency () const

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold () const

  Actions further into the future than this are early.
- uint16_t getCapacity () const

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull () const

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate () const

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount () const

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount () const

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount () const

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount () const

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount () const

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount () const

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)
- void **compile** ()
- const std::string & **getObjectName** () const
- const std::string & **getObjectPath** () const
- const Conditions & **getConditions** () const
- unsigned **getChannel** () const
- unsigned **getNum** () const
- virtual void **receiveMSI** (uint8_t code)
- Condition * **getCondition** (const std::string object_path)
- void **removeCondition** (Condition *condition)
- unsigned **createConditionNumber** ()
- template<typename ConditionType , typename... Args>
  std::string **NewConditionHelper** (bool active, Args &&... args)

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Additional Inherited Members

Public Types inherited from saftlib::ActionSink

- typedef std::map< unsigned, std::unique_ptr< Condition > > **Conditions**

Public Attributes inherited from saftlib::ActionSink

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::ActionSink

- Record **fetchError** (uint8_t code) const
- bool **updateOverflow** () const
- bool **updateAction** () const
- bool **updateLate** () const
- bool **updateEarly** () const
- bool **updateConflict** () const
- bool **updateDelayed** () const

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes inherited from saftlib::ActionSink

- std::string **object_path**
- ECA & **eca**
- std::string **name**
- unsigned **channel**
- unsigned **num**
- int64_t **minOffset**
- int64_t **maxOffset**
- std::chrono::nanoseconds **signalRate**
- uint64_t **overflowCount**
- uint64_t **actionCount**
- uint64_t **lateCount**
- uint64_t **earlyCount**
- uint64_t **conflictCount**
- uint64_t **delayedCount**
- std::chrono::steady_clock::time_point **overflowUpdate**
- std::chrono::steady_clock::time_point **actionUpdate**
- std::chrono::steady_clock::time_point **lateUpdate**
- std::chrono::steady_clock::time_point **earlyUpdate**
- std::chrono::steady_clock::time_point **conflictUpdate**
- std::chrono::steady_clock::time_point **delayedUpdate**
- uint64_t **latency**
- uint64_t **earlyThreshold**
- uint16_t **capacity**
- saftbus::SourceHandle **overflowPending**
- saftbus::SourceHandle **actionPending**
- saftbus::SourceHandle **latePending**
- saftbus::SourceHandle **earlyPending**
- saftbus::SourceHandle **conflictPending**
- saftbus::SourceHandle **delayedPending**
- Conditions **conditions**
- saftbus::Container * **container**

7.44.1 Detailed Description

An output through which EmbeddedCPU actions flow.

de.gsi.saftlib.EmbeddedCPUActionSink:

This inteface allows the generation of SCU timing events. A EmbeddedCPUActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.44.2 Member Function Documentation

7.44.2.1 NewCondition()

```
std::string saftlib::EmbeddedCPUActionSink::NewCondition (
            bool active,
            uint64_t id,
            uint64_t mask,
            int64_t offset,
            uint32_t tag )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
tag	The 32-bit value to send on the SCUbus
result	Object path to the created SCUbusCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ∼mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SCUBUSCondition object.

The documentation for this class was generated from the following files:

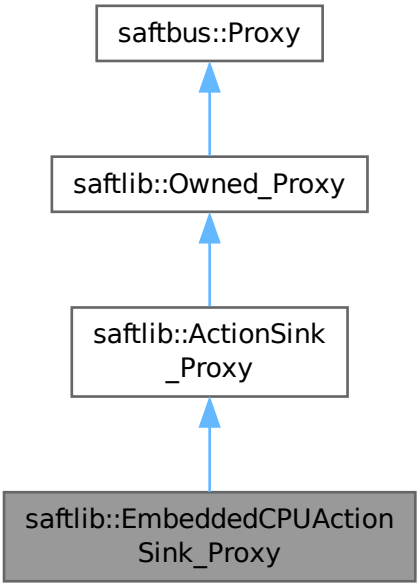
- src/EmbeddedCPUActionSink.hpp
- src/EmbeddedCPUActionSink.cpp

7.45 saftlib::EmbeddedCPUActionSink_Proxy Class Reference

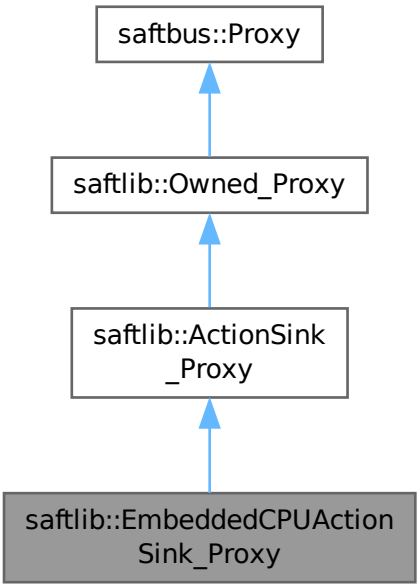
An output through which EmbeddedCPU actions flow.

```
#include <EmbeddedCPUActionSink_Proxy.hpp>
```

Inheritance diagram for saftlib::EmbeddedCPUActionSink_Proxy:



Collaboration diagram for saftlib::EmbeddedCPUActionSink_Proxy:



Public Member Functions

- **EmbeddedCPUActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_↩ group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_↩ interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag)

  Create a condition to match incoming events.

Public Member Functions inherited from saftlib::ActionSink_Proxy

- **ActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions ()

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions ()

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions ()

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset ()

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset ()

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency ()

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold ()

  Actions further into the future than this are early.
- uint16_t getCapacity ()

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull ()

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate ()

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount ()

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount ()

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)

- uint64_t getLateCount ()

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount ()

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount ()

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount ()

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< EmbeddedCPUActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ActionSink_Proxy

- static std::shared_ptr< ActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::ActionSink_Proxy

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.45.1 Detailed Description

An output through which EmbeddedCPU actions flow.

de.gsi.saftlib.EmbeddedCPUActionSink:

This inteface allows the generation of SCU timing events. A EmbeddedCPUActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.45.2 Member Function Documentation

7.45.2.1 NewCondition()

```
std::string saftlib::EmbeddedCPUActionSink_Proxy::NewCondition (
            bool active,
            uint64_t id,
            uint64_t mask,
            int64_t offset,
            uint32_t tag )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
tag	The 32-bit value to send on the SCUbus
result	Object path to the created SCUbusCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ∼mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SCUBUSCondition object.

7.45.2.2 signal_dispatch()

```
bool saftlib::EmbeddedCPUActionSink_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

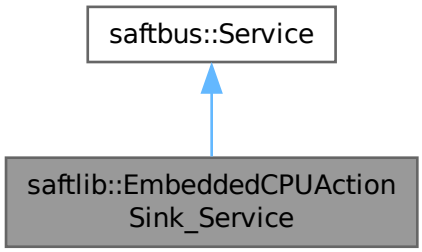
Reimplemented from saftlib::ActionSink_Proxy.

The documentation for this class was generated from the following files:

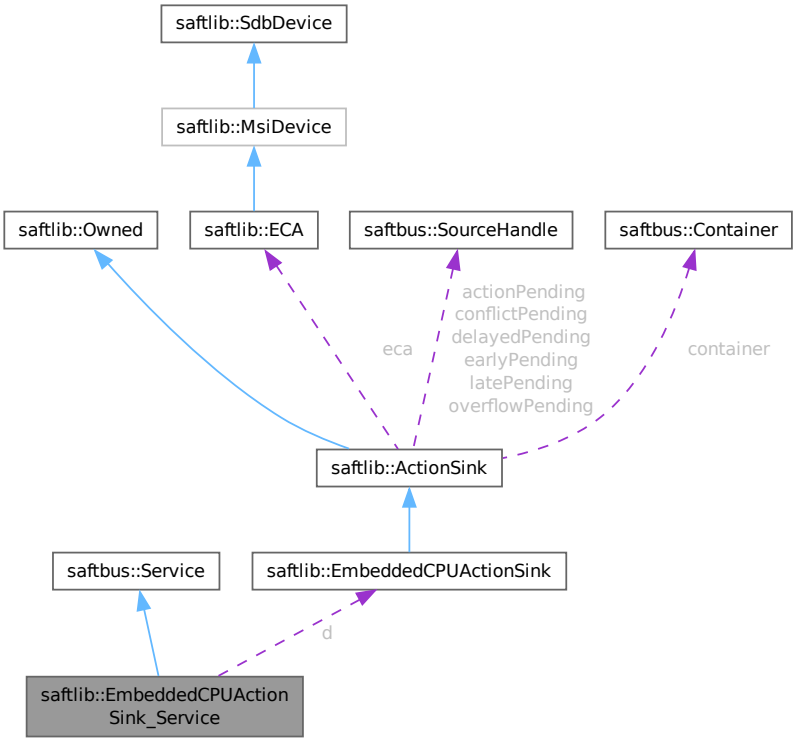
- src/EmbeddedCPUActionSink_Proxy.hpp
- src/EmbeddedCPUActionSink_Proxy.cpp

7.46 saftlib::EmbeddedCPUActionSink_Service Class Reference

Inheritance diagram for saftlib::EmbeddedCPUActionSink_Service:



Collaboration diagram for saftlib::EmbeddedCPUActionSink_Service:



Public Types

- typedef EmbeddedCPUActionSink **DriverType**

Public Member Functions

- **EmbeddedCPUActionSink_Service** (EmbeddedCPUActionSink *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **OverflowCount_dispatch_function** (uint64_t arg_1)
- void **ActionCount_dispatch_function** (uint64_t arg_1)
- void **LateCount_dispatch_function** (uint64_t arg_1)
- void **SigLate_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_↩ 4, saftlib::Time arg_5)
- void **EarlyCount_dispatch_function** (uint64_t arg_1)
- void **SigEarly_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_↩ 4, saftlib::Time arg_5)
- void **ConflictCount_dispatch_function** (uint64_t arg_1)
- void **SigConflict_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **DelayedCount_dispatch_function** (uint64_t arg_1)
- void **SigDelayed_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- EmbeddedCPUActionSink * **d**
- sigc::connection **OverflowCount_connection**
- sigc::connection **ActionCount_connection**
- sigc::connection **LateCount_connection**
- sigc::connection **SigLate_connection**
- sigc::connection **EarlyCount_connection**
- sigc::connection **SigEarly_connection**
- sigc::connection **ConflictCount_connection**
- sigc::connection **SigConflict_connection**
- sigc::connection **DelayedCount_connection**
- sigc::connection **SigDelayed_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.46.1 Member Function Documentation

7.46.1.1 call()

```
void saftlib::EmbeddedCPUActionSink_Service::call (
             unsigned interface_no,
             unsigned function_no,
             int client_fd,
             saftbus::Deserializer & received,
             saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

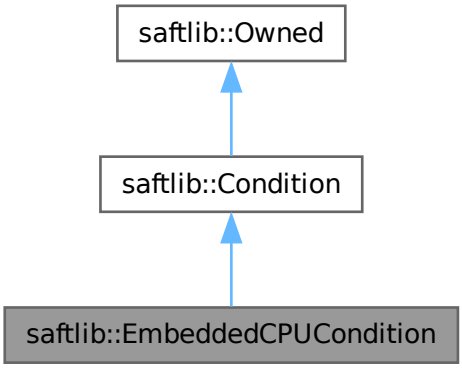
- src/EmbeddedCPUActionSink_Service.hpp
- src/EmbeddedCPUActionSink_Service.cpp

7.47 saftlib::EmbeddedCPUCondition Class Reference

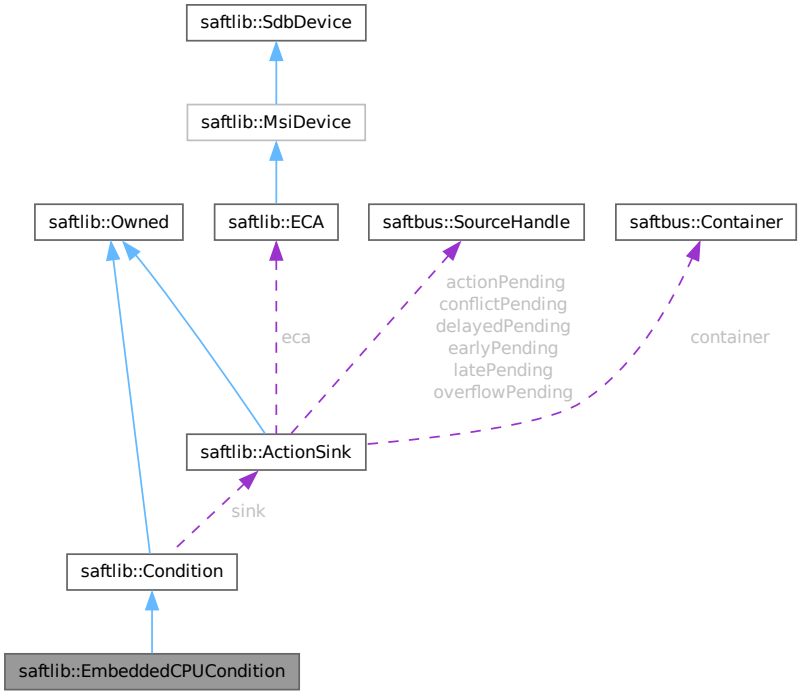
Matched against incoming events on a EmbeddedCPUActionSink.

```
#include <EmbeddedCPUCondition.hpp>
```

Inheritance diagram for saftlib::EmbeddedCPUCondition:



Collaboration diagram for saftlib::EmbeddedCPUCondition:



Public Types

- typedef EmbeddedCPUCondition_Service **ServiceType**

Public Member Functions

- **EmbeddedCPUCondition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)
- uint32_t getTag () const

  The tag which is sent to the SCUbus by this condition.
- void setTag (uint32_t val)

  The tag which is sent to the SCUbus by this condition.

Public Member Functions inherited from saftlib::Condition

- **Condition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)
- uint64_t getID () const

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask () const

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset () const

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate () const

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly () const

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict () const

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed () const

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive () const

  The condition should be actively matched against events.
- void **setActive** (bool val)
- std::string & **getObjectPath** ()
- uint32_t **getRawTag** () const
- void **setRawActive** (bool val)
- unsigned **getNumber** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)

- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes inherited from saftlib::Condition

- std::string **objectPath**
- ActionSink * **sink**
- unsigned **number**
- uint64_t **id**
- uint64_t **mask**
- int64_t **offset**
- uint32_t **tag**
- bool **acceptLate**
- bool **acceptEarly**
- bool **acceptConflict**
- bool **acceptDelayed**
- bool **active**

7.47.1 Detailed Description

Matched against incoming events on a EmbeddedCPUActionSink.

de.gsi.saftlib.EmbeddedCPUCondition:

EmbeddedCPUConditions are created by EmbeddedCPUActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

7.47.2 Member Function Documentation

7.47.2.1 getTag()

```
uint32_t saftlib::EmbeddedCPUCondition::getTag ( ) const
```

The tag which is sent to the SCUbus by this condition.

Returns

The tag value

7.47.2.2 setTag()

```
void saftlib::EmbeddedCPUCondition::setTag (
            uint32_t val )
```

The tag which is sent to the SCUbus by this condition.

Parameters

val	The tag value

The documentation for this class was generated from the following files:

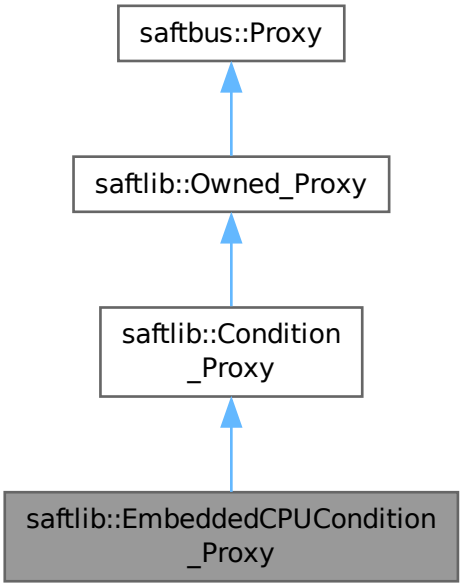
- src/EmbeddedCPUCondition.hpp
- src/EmbeddedCPUCondition.cpp

7.48 saftlib::EmbeddedCPUCondition_Proxy Class Reference

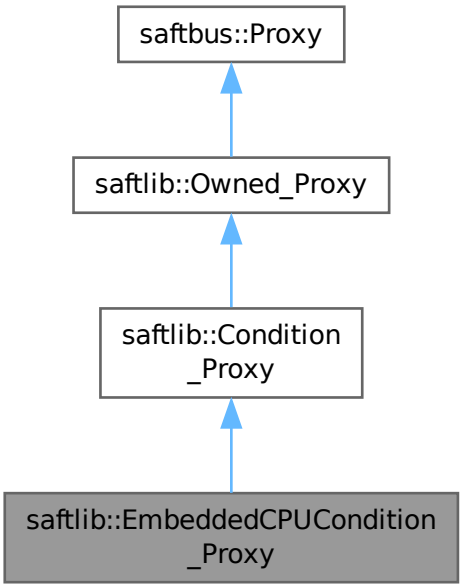
Matched against incoming events on a EmbeddedCPUActionSink.

```
#include <EmbeddedCPUCondition_Proxy.hpp>
```

Inheritance diagram for saftlib::EmbeddedCPUCondition_Proxy:



Collaboration diagram for saftlib::EmbeddedCPUCondition_Proxy:



Public Member Functions

- **EmbeddedCPUCondition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_↩ group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_↩ interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint32_t getTag ()

  The tag which is sent to the SCUbus by this condition.
- void setTag (uint32_t val)

  The tag which is sent to the SCUbus by this condition.

Public Member Functions inherited from saftlib::Condition_Proxy

- **Condition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t getID ()

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask ()

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset ()

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate ()

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly ()

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict ()

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed ()

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive ()

  The condition should be actively matched against events.
- void **setActive** (bool val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< EmbeddedCPUCondition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Condition_Proxy

- static std::shared_ptr< Condition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.48.1 Detailed Description

Matched against incoming events on a EmbeddedCPUActionSink.

de.gsi.saftlib.EmbeddedCPUCondition:

EmbeddedCPUConditions are created by EmbeddedCPUActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

7.48.2 Member Function Documentation

7.48.2.1 getTag()

```
uint32_t saftlib::EmbeddedCPUCondition_Proxy::getTag ( )
```

The tag which is sent to the SCUbus by this condition.

Returns

The tag value

7.48.2.2 setTag()

```
void saftlib::EmbeddedCPUCondition_Proxy::setTag (
            uint32_t val )
```

The tag which is sent to the SCUbus by this condition.

Parameters

val	The tag value

7.48.2.3 signal_dispatch()

```
bool saftlib::EmbeddedCPUCondition_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

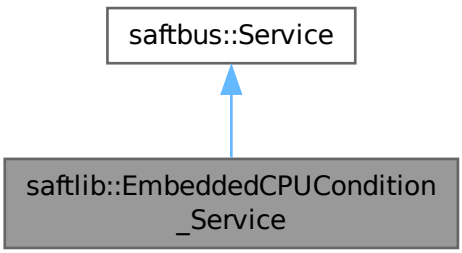
Reimplemented from saftlib::Condition_Proxy.

The documentation for this class was generated from the following files:

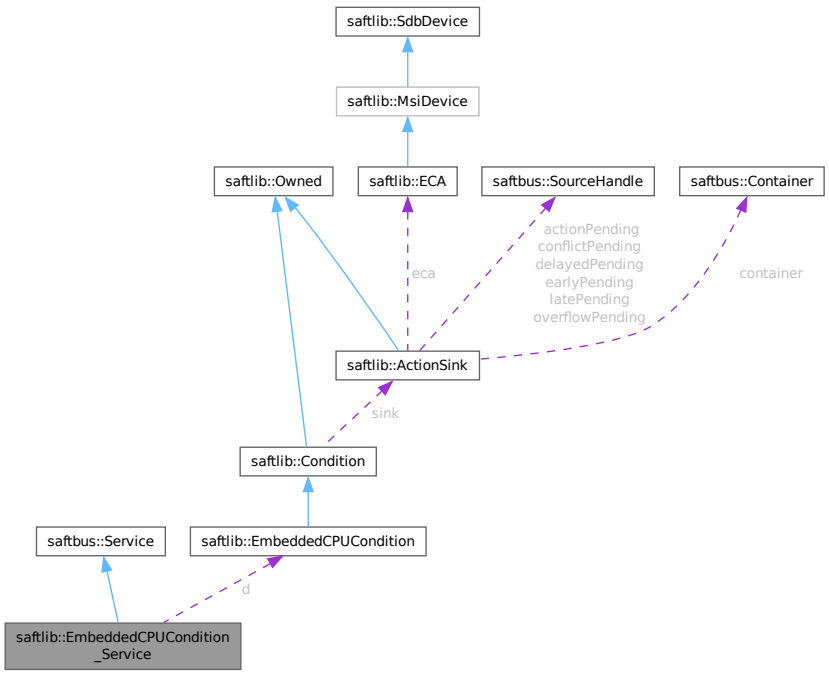
- src/EmbeddedCPUCondition_Proxy.hpp
- src/EmbeddedCPUCondition_Proxy.cpp

7.49 saftlib::EmbeddedCPUCondition_Service Class Reference

Inheritance diagram for saftlib::EmbeddedCPUCondition_Service:



Collaboration diagram for saftlib::EmbeddedCPUCondition_Service:



Public Types

- typedef EmbeddedCPUCondition **DriverType**

Public Member Functions

- **EmbeddedCPUCondition_Service** (EmbeddedCPUCondition *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- EmbeddedCPUCondition * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.49.1 Member Function Documentation

7.49.1.1 call()

```
void saftlib::EmbeddedCPUCondition_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

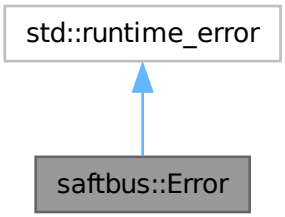
Implements saftbus::Service.

The documentation for this class was generated from the following files:

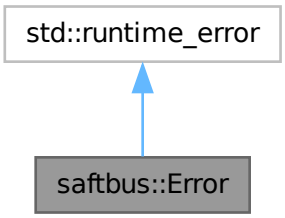
- src/EmbeddedCPUCondition_Service.hpp
- src/EmbeddedCPUCondition_Service.cpp

7.50 saftbus::Error Class Reference

Inheritance diagram for saftbus::Error:



Collaboration diagram for saftbus::Error:



Public Types

- enum **Type** {
  NO_ERROR , **INVALID_ARGS** , **UNKNOWN_METHOD** , **IO_ERROR** ,
  ACCESS_DENIED , **FAILED** }

Public Member Functions

- **Error** (Type type, const std::string &msg)
- **Error** (const std::string &msg)
- **Error** (const Error &error)
- Type **type** () const

The documentation for this class was generated from the following files:

- saftbus/error.hpp
- saftbus/error.cpp

7.51 software_tr::SoftwareECA::Event Struct Reference

Public Member Functions

- **Event** (uint64_t _id, uint64_t _param, uint64_t _deadline, uint32_t _num, int32_t _tag, uint32_t _msi_adr, uint32_t _msi_dat)
- bool **operator==** (const Event &rhs) const
- bool **operator<** (const Event &rhs) const

Public Attributes

- uint64_t **id**
- uint64_t **param**
- uint64_t **deadline**
- uint32_t **num**
- int32_t **tag**
- uint32_t **msi_adr**
- uint32_t **msi_dat**

The documentation for this struct was generated from the following file:

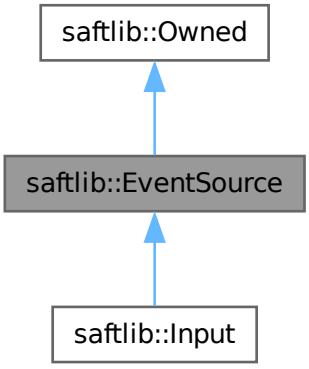
- src/saft-software-tr.cpp

7.52 saftlib::EventSource Class Reference

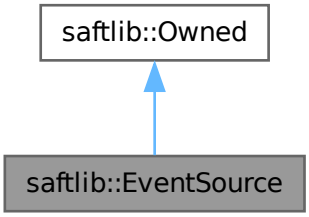
Interface of ECA event sources currently only implemented by saftlib::Input.

`#include <EventSource.hpp>`

Inheritance diagram for saftlib::EventSource:



Collaboration diagram for saftlib::EventSource:



Public Member Functions

- **EventSource** (const std::string &object_path, const std::string &name, saftbus::Container *container)
- virtual uint64_t getResolution () const =0

  The precision of generated timestamps in nanoseconds.
- virtual uint32_t getEventBits () const =0

  How many bits of external data are included in the ID.
- virtual bool getEventEnable () const =0

  Should the event source generate events.

- virtual void **setEventEnable** (bool val)=0
- virtual uint64_t getEventPrefix () const =0

  Combined with low EventBits to create generated IDs.
- virtual void **setEventPrefix** (uint64_t val)=0
- std::string **getObjectPath** () const
- std::string **getObjectName** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container ∗container)
- void **set_service** (saftbus::Service ∗service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

7.52.1 Detailed Description

Interface of ECA event sources currently only implemented by saftlib::Input.

7.52.2 Member Function Documentation

7.52.2.1 getEventBits()

```
virtual uint32_t saftlib::EventSource::getEventBits ( ) const  [pure virtual]
```

How many bits of external data are included in the ID.

Returns

How many bits of external data are included in the ID

Implemented in saftlib::Input.

7.52.2.2 getEventEnable()

```
virtual bool saftlib::EventSource::getEventEnable ( ) const  [pure virtual]
```

Should the event source generate events.

Returns

true if the event source generates events

Implemented in saftlib::Input.

7.52.2.3 getEventPrefix()

```
virtual uint64_t saftlib::EventSource::getEventPrefix ( ) const  [pure virtual]
```

Combined with low EventBits to create generated IDs.

Returns

the event prefix

Implemented in saftlib::Input.

7.52.2.4 getResolution()

```
virtual uint64_t saftlib::EventSource::getResolution ( ) const  [pure virtual]
```

The precision of generated timestamps in nanoseconds.

Returns

The precision of generated timestamps in nanoseconds.

Implemented in saftlib::Input.

The documentation for this class was generated from the following files:

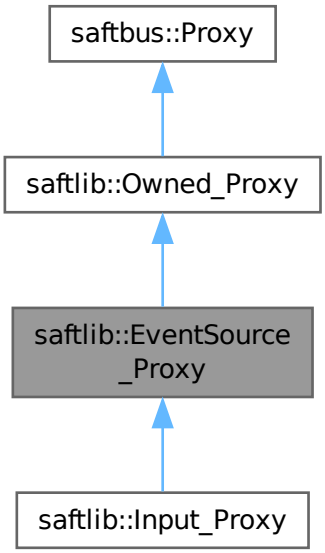
- src/EventSource.hpp
- src/EventSource.cpp

7.53 saftlib::EventSource_Proxy Class Reference

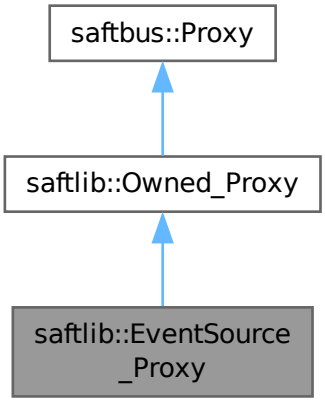
Interface of ECA event sources currently only implemented by saftlib::Input.

`#include <EventSource_Proxy.hpp>`

Inheritance diagram for saftlib::EventSource_Proxy:



Collaboration diagram for saftlib::EventSource_Proxy:



Public Member Functions

- **EventSource_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- virtual uint64_t getResolution ()

  The precision of generated timestamps in nanoseconds.
- virtual uint32_t getEventBits ()

  How many bits of external data are included in the ID.
- virtual bool getEventEnable ()

  Should the event source generate events.
- virtual void **setEventEnable** (bool val)
- virtual uint64_t getEventPrefix ()

  Combined with low EventBits to create generated IDs.
- virtual void **setEventPrefix** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< EventSource_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.53.1 Detailed Description

Interface of ECA event sources currently only implemented by saftlib::Input.

7.53.2 Member Function Documentation

7.53.2.1 getEventBits()

```
uint32_t saftlib::EventSource_Proxy::getEventBits ( )  [virtual]
```

How many bits of external data are included in the ID.

Returns

How many bits of external data are included in the ID

7.53.2.2 getEventEnable()

```
bool saftlib::EventSource_Proxy::getEventEnable ( )  [virtual]
```

Should the event source generate events.

Returns

true if the event source generates events

7.53.2.3 getEventPrefix()

```
uint64_t saftlib::EventSource_Proxy::getEventPrefix ( )  [virtual]
```

Combined with low EventBits to create generated IDs.

Returns

the event prefix

7.53.2.4 getResolution()

```
uint64_t saftlib::EventSource_Proxy::getResolution ( )  [virtual]
```

The precision of generated timestamps in nanoseconds.

Returns

The precision of generated timestamps in nanoseconds.

7.53.2.5 signal_dispatch()

```
bool saftlib::EventSource_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

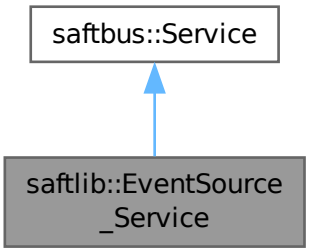
Reimplemented in saftlib::Input_Proxy.

The documentation for this class was generated from the following files:

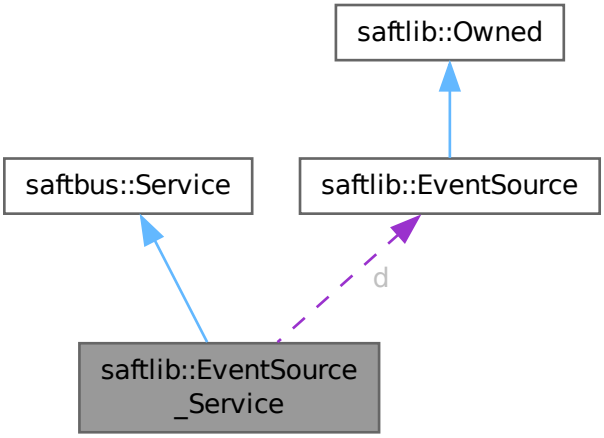
- src/EventSource_Proxy.hpp
- src/EventSource_Proxy.cpp

7.54 saftlib::EventSource_Service Class Reference

Inheritance diagram for saftlib::EventSource_Service:



Collaboration diagram for saftlib::EventSource_Service:



Public Types

- typedef EventSource **DriverType**

Public Member Functions

- **EventSource_Service** (EventSource *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- EventSource * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.54.1 Member Function Documentation

7.54.1.1 call()

```
void saftlib::EventSource_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

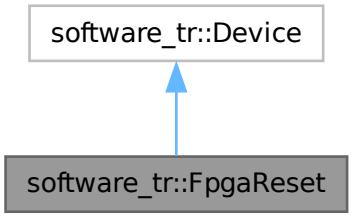
Implements saftbus::Service.

The documentation for this class was generated from the following files:

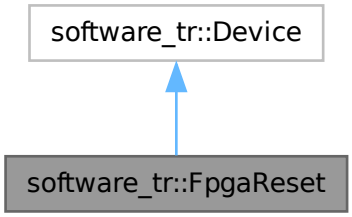
- src/EventSource_Service.hpp
- src/EventSource_Service.cpp

7.55 software_tr::FpgaReset Class Reference

Inheritance diagram for software_tr::FpgaReset:



Collaboration diagram for software_tr::FpgaReset:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0x3a362063 }

Public Member Functions

- **FpgaReset** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool write_access (uint32_t adr, int sel, uint32_t dat)
- bool read_access (uint32_t adr, int sel, uint32_t ∗dat_out)

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t ∗dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

Static Public Attributes

- static bool **_reset_was_triggered** = false

7.55.1 Member Function Documentation

7.55.1.1 contains()

```
bool software_tr::FpgaReset::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.55.1.2 read_access()

```
bool software_tr::FpgaReset::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

7.55.1.3 write_access()

```
bool software_tr::FpgaReset::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

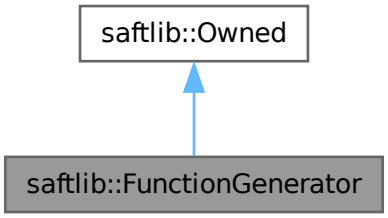
- src/saft-software-tr.cpp

7.56 saftlib::FunctionGenerator Class Reference

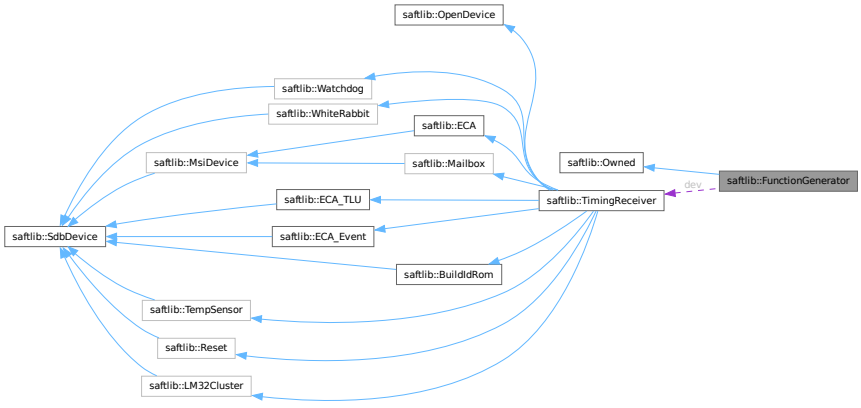
Function Generator for creating timing triggered waveforms.

`#include <FunctionGenerator.hpp>`

Inheritance diagram for saftlib::FunctionGenerator:



Collaboration diagram for saftlib::FunctionGenerator:



Public Member Functions

- **FunctionGenerator** (saftbus::Container *container, const std::string &fg_name, const std::string &object_path, std::shared_ptr< FunctionGeneratorImpl > impl)
- void Arm ()

  Enable the function generator and arm it.
- void Abort ()

  Abort waveform generation.
- uint64_t ReadFillLevel ()

  Remaining waveform data in nanoseconds.
- bool AppendParameterSet (const std::vector< int16_t > &coeff_a, const std::vector< int16_t > &coeff_b, const std::vector< int32_t > &coeff_c, const std::vector< unsigned char > &step, const std::vector< unsigned char > &freq, const std::vector< unsigned char > &shift_a, const std::vector< unsigned char > &shift_b)

  Append parameter tuples describing waveform to generate.
- void Flush ()

  Empty the parameter tuple set.
- uint32_t getVersion () const

  Version of the hardware macro.
- unsigned char getSCUbusSlot () const

  Slot number of the slave card.
- unsigned char getDeviceNumber () const

  number of the hardware macro inside of the slave card
- unsigned char getOutputWindowSize () const

  Number of bits output by the function generator.
- bool getEnabled () const

  Hardware is allowed to generate an output waveform. If a function generator is disabled, its output is constant. Enabled is set to true using Arm and transitions to false either upon completion or after a call to Abort. SigEnabled is emitted when this changes.
- bool getArmed () const

  Upon receipt of StartTag, output will begin.
- bool getRunning () const

  The function generator is currently producing a waveform.
- uint32_t getStartTag () const

  The SCUbus tag which causes function generation to begin.
- void **setStartTag** (uint32_t val)
- uint32_t ReadExecutedParameterCount ()

  Number of parameter tuples executed by hardware.
- std::string **getObjectPath** ()

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

Claim ownership of the object.

- std::string getOwner () const

  The client which owns this object.

- bool getDestructible () const

  Can the object be destroyed.

- void Destroy ()

  Destroy this object.

Public Attributes

- sigc::signal< void, bool > **SigEnabled**

  Hardware is allowed to generate an output waveform. If a function generator is disabled, its output is constant. Enabled is set to true using Arm and transitions to false either upon completion or after a call to Abort. The current state of Enabled can be obtained using getEnabled().

- sigc::signal< void, bool > **SigArmed**

  Upon receipt of StartTag, output will begin. If a function generator is Armed, it is also Enabled. Once waveform data has been loaded into the function generator, Arm it. Shortly thereafter (once the hardware is ready), the Armed property will transition to true, indicating the hardware is ready to react. Once StartTag is received, Armed changes to false and Started is emitted. getArmed() method can be used to obtain the current state.

- sigc::signal< void, bool > SigRunning

  The function generator is currently producing a waveform.

- sigc::signal< void, saftlib::Time > SigStarted

  Function generation has begun.

- sigc::signal< void, saftlib::Time, bool, bool, bool > SigStopped

  Function generation has ended.

- sigc::signal< void > Refill

  More tuples must be appended to ensure uninterrupted waveform.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions

- void **Reset** ()
- void **ownerQuit** ()
- void **on_fg_running** (bool)
- void **on_fg_armed** (bool)
- void **on_fg_enabled** (bool)
- void **on_fg_refill** ()
- void **on_fg_started** (uint64_t)
- void **on_fg_stopped** (uint64_t, bool, bool, bool)

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes

- TimingReceiver * **dev**
- std::string **name**
- std::string **objectPath**
- std::shared_ptr< FunctionGeneratorImpl > **fgImpl**

7.56.1 Detailed Description

Function Generator for creating timing triggered waveforms.

The function generator creates a waveform from a piecewise continuous sequence of second order polynomials. These polynomials are specified using a fixed-width ¨parameter tuple¨ format. Once triggered, the function generator outputs samples created from each tupled polynomial until it exhausts the specified piecewise width of the polynomial, whereupon it begins outputting samples from the next tupled polynomial.

The general work-flow to use the FunctionGenerator is to append sets of tuples describing the waveform using the AppendParameterSet function. Then, select a SCUbus timing tag (StartTag) whose appearance should trigger waveform generation. Finally, the FunctionGenerator is armed by calling Arm, whereafter it may be triggered manually via the SCUbusActionSink->InjectTag method, or by a timing event sent by the data master, which is configured to create the StartTag.

If you need an unending waveform, follow the steps above, but additionally monitor the Refill signal, and append additional parameter tuples until low_fill is false.

The sequence of signals of successful waveform generation occur in this order: Enabled(true) Armed(true) Armed(false) Started Running(true) Refill Running(false) Stopped Enabled(false)

7.56.2 Member Function Documentation

7.56.2.1 Abort()

```
void saftlib::FunctionGenerator::Abort ( )
```

Abort waveform generation.

This directs the hardware to stop waveform generation. If the function generator was Armed, it is disarmed and disabled, without outputting any waveform data. If the function generator is running, output is Stopped at the current value and disabled. Aborting a function generator takes time, so even after a call to Abort, the function generator might still be Started. However, it will reach the disabled state as quickly as it can, transitioning through Stopped as usual. If the Owner of a FunctionGenerator quits without running Disown, the Abort is run automatically.

7.56.2.2 AppendParameterSet()

```
bool saftlib::FunctionGenerator::AppendParameterSet (
            const std::vector< int16_t > & coeff_a,
            const std::vector< int16_t > & coeff_b,
            const std::vector< int32_t > & coeff_c,
            const std::vector< unsigned char > & step,
            const std::vector< unsigned char > & freq,
            const std::vector< unsigned char > & shift_a,
            const std::vector< unsigned char > & shift_b )
```

Append parameter tuples describing waveform to generate.

Parameters

coeff_a	Quadratic coefficient (a), 16-bit signed
coeff_b	Linear coefficient (b), 16-bit signed
coeff_c	Constant coefficient (c), 32-bit signed
step	Number of interpolated samples (0=250, 1=500, 2=1000, 3=2000, 4=4000, 5=8000, 6=16000, or 7=32000)
freq	Output sample frequency (0=16kHz, 1=32kHz, 2=64kHz, 3=125kHz, 4=250kHz, 5=500kHz, 6=1MHz, or 7=2MHz)
shift_a	Exponent of coeff_a, 6-bit unsigned; a = coeff_a∗2^shift_a
shift_b	Exponent of coeff_b, 6-bit unsigned; b = coeff_b∗2^shift_b
low_fill	Fill level remains too low

This function appends the parameter vectors (which must be equal in length) to the FIFO of remaining waveform to generate. Each parameter set (coefficients) describes a number of output samples in the generated wave form. Parameter sets are executed in order until no more remain.

If the fill level is not high enough, this method returns true. Only once this function has returned false can you await the Refill signal.

At each step, the function generator outputs high_bits(c∗2^32 + b∗t + c∗t∗t), where t ranges from 0 to numSteps-1. high_bits are the high OutputWindowSize bits of the resulting 64-bit signed value.

7.56.2.3 Arm()

```
void saftlib::FunctionGenerator::Arm ( )
```

Enable the function generator and arm it.

A function generator can only be Armed if FillLevel is non-zero. An Enabled function generator can not be Armed again until it either completes waveform generation or the user calls Abort. Arming a function generator takes time. Wait for Armed to transition to true before sending StartTag.

7.56.2.4 Flush()

```
void saftlib::FunctionGenerator::Flush ( )
```

Empty the parameter tuple set.

Flush may only be called when not Enabled. Flush does not clear the ExecutedParameterCount.

7.56.2.5 getArmed()

```
bool saftlib::FunctionGenerator::getArmed ( ) const
```

Upon receipt of StartTag, output will begin.

If a function generator is Armed, it is also Enabled. Once waveform data has been loaded into the function generator, Arm it. Shortly thereafter (once the hardware is ready), the Armed property will transition to true, indicating the hardware is ready to react. Once StartTag is received, Armed changes to false and Started is emitted. SigArmed is emitted when this changes.

Returns

true if armed

7.56.2.6 getDeviceNumber()

```
unsigned char saftlib::FunctionGenerator::getDeviceNumber ( ) const
```

number of the hardware macro inside of the slave card

Returns

number of the hardware macro inside of the slave card

7.56.2.7 getEnabled()

```
bool saftlib::FunctionGenerator::getEnabled ( ) const
```

Hardware is allowed to generate an output waveform. If a function generator is disabled, its output is constant. Enabled is set to true using Arm and transitions to false either upon completion or after a call to Abort. SigEnabled is emitted when this changes.

Returns

true if enabled

7.56.2.8 getOutputWindowSize()

```
unsigned char saftlib::FunctionGenerator::getOutputWindowSize ( ) const
```

Number of bits output by the function generator.

Returns

Number of bits output by the function generator

7.56.2.9 getRunning()

```
bool saftlib::FunctionGenerator::getRunning ( ) const
```

The function generator is currently producing a waveform.

Function generation is started by sending a SCUbus tag which matches the StartTag property of an Armed function generator. When this property transitions to true, the Started signal is emitted. When this property transitions to false, the Stopped signal is emitted. SigRunning or is emitted when this changes.

Returns

true if running

7.56.2.10 getSCUbusSlot()

```
unsigned char saftlib::FunctionGenerator::getSCUbusSlot ( ) const
```

Slot number of the slave card.

Returns

Slot number of the slave card

7.56.2.11 getStartTag()

```
uint32_t saftlib::FunctionGenerator::getStartTag ( ) const
```

The SCUbus tag which causes function generation to begin.

If the function generator is Armed and this tag is sent to the SCUbus, then the function generator will begin generating the output waveform. StartTag may only be set when the FunctionGenerator is not Enabled.

Returns

SCUbus tag

7.56.2.12 getVersion()

```
uint32_t saftlib::FunctionGenerator::getVersion ( ) const
```

Version of the hardware macro.

Returns

Version of the hardware macro

7.56.2.13 ReadExecutedParameterCount()

```
uint32_t saftlib::FunctionGenerator::ReadExecutedParameterCount ( )
```

Number of parameter tuples executed by hardware.

This counts the total number of parameter tuples executed since the last Started signal. Obviously, if the function generator is running, the returned value will be old.

Returns

Number tuples executed by hardware.

7.56.2.14 ReadFillLevel()

```
uint64_t saftlib::FunctionGenerator::ReadFillLevel ( )
```

Remaining waveform data in nanoseconds.

The SAFTd has sufficient parameters buffered to supply the function generator with data for the specified time in nanoseconds. Note, due to the slow nature of software, if the function generator is currently running, the read value will already be out-of-date upon return. This property should be used for informational use only.

Returns

Remaining waveform data in nanoseconds.

7.56.3 Member Data Documentation

7.56.3.1 Refill

```
sigc::signal< void > saftlib::FunctionGenerator::Refill
```

More tuples must be appended to ensure uninterrupted waveform.

In order to guarantee an uninterrupted supply of data to the function generator, there should be data buffered in the SAFTd. When the buffer fill level gets too low, this signal is emitted, and you should run AppendParameterSet. If you do not, function generation will cease, signalling successful completion of the waveform with Stopped.

7.56.3.2 SigRunning

```
sigc::signal< void , bool > saftlib::FunctionGenerator::SigRunning
```

The function generator is currently producing a waveform.

Function generation is started by sending a SCUbus tag which matches the StartTag property of an Armed function generator. When this property transitions to true, the Started signal is emitted. When this property transitions to false, the Stopped signal is emitted. getRunning() method can be used to obtain the current state.

7.56.3.3 SigStarted

```
sigc::signal< void , saftlib::Time > saftlib::FunctionGenerator::SigStarted
```

Function generation has begun.

This signal notifies software when function generation has begun, possibly to update a GUI or other user-facing status information.

Parameters

time	Time when function generation began in nanoseconds since 1970

7.56.3.4 SigStopped

`sigc::signal< void , saftlib::Time , bool , bool , bool > saftlib::FunctionGenerator::SigStopped`

Function generation has ended.

Parameters

time	Time when function generation ended in nanoseconds since 1970
abort	stopped due to a call to Abort
hardwareMacroUnderflow	A fatal error, indicating the SCUbus is congested
microControllerUnderflow	A fatal error, indicating the host CPU is overloaded

The function generator stops either successfully (when all data has been sent), or it stops due to an error. When an error occurs, the function generator stops and holds its most recent value. This can occur due to two causes:

hardwareMacroUnderflow, a fatal error indicating the hardware ran out of data. If the SCUbus is too busy, it can happen that the waveform data stored in the function generator HDL is not refilled in time. This error can only be mitigated by ensuring that the function generator does not share the SCUbus with other users.

microControllerUnderflow, a fatal error indicating the microcontroller ran out of data. If the host CPU running this software is too busy, it can happen that the waveform data is not delivered to the microcontroller before the microcontroller runs out of data. This error can be mitigated by reducing the number of busy processes running on the system.

Once the function generator has stopped, ExecutedParameterCount remains valid until the next time the function generator starts. After stopping, regardless of if the generation was successful or not, the parameter FIFO is cleared, Enabled is false, and this signal emitted.

The documentation for this class was generated from the following files:

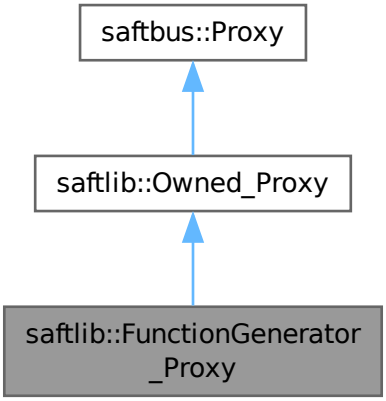
- src/FunctionGenerator.hpp
- src/FunctionGenerator.cpp

7.57 saftlib::FunctionGenerator_Proxy Class Reference

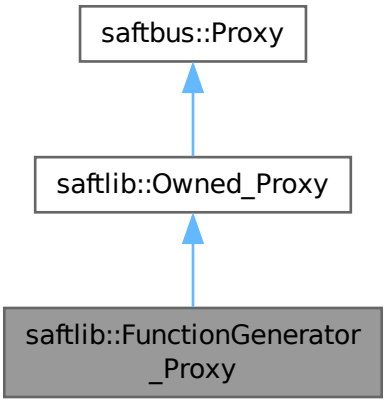
Function Generator for creating timing triggered waveforms.

`#include <FunctionGenerator_Proxy.hpp>`

Inheritance diagram for saftlib::FunctionGenerator_Proxy:



Collaboration diagram for saftlib::FunctionGenerator_Proxy:



Public Member Functions

- **FunctionGenerator_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::↩ SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Arm ()

  Enable the function generator and arm it.
- void Abort ()

Abort waveform generation.

- uint64_t ReadFillLevel ()

  Remaining waveform data in nanoseconds.

- bool AppendParameterSet (const std::vector< int16_t > &coeff_a, const std::vector< int16_t > &coeff_b, const std::vector< int32_t > &coeff_c, const std::vector< unsigned char > &step, const std::vector< unsigned char > &freq, const std::vector< unsigned char > &shift_a, const std::vector< unsigned char > &shift_b)

  Append parameter tuples describing waveform to generate.

- void Flush ()

  Empty the parameter tuple set.

- uint32_t getVersion ()

  Version of the hardware macro.

- unsigned char getSCUbusSlot ()

  Slot number of the slave card.

- unsigned char getDeviceNumber ()

  number of the hardware macro inside of the slave card

- unsigned char getOutputWindowSize ()

  Number of bits output by the function generator.

- bool getEnabled ()

  Hardware is allowed to generate an output waveform. If a function generator is disabled, its output is constant. Enabled is set to true using Arm and transitions to false either upon completion or after a call to Abort. SigEnabled is emitted when this changes.

- bool getArmed ()

  Upon receipt of StartTag, output will begin.

- bool getRunning ()

  The function generator is currently producing a waveform.

- uint32_t getStartTag ()

  The SCUbus tag which causes function generation to begin.

- void **setStartTag** (uint32_t val)
- uint32_t ReadExecutedParameterCount ()

  Number of parameter tuples executed by hardware.

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

- void Disown ()

  Release ownership of the object.

- void Own ()

  Claim ownership of the object.

- std::string getOwner ()

  The client which owns this object.

- bool getDestructible ()

  Can the object be destroyed.

- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< FunctionGenerator_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Public Attributes

- sigc::signal< void, bool > **SigEnabled**

  Hardware is allowed to generate an output waveform. If a function generator is disabled, its output is constant. Enabled is set to true using Arm and transitions to false either upon completion or after a call to Abort. The current state of Enabled can be obtained using getEnabled().
- sigc::signal< void, bool > **SigArmed**

  Upon receipt of StartTag, output will begin. If a function generator is Armed, it is also Enabled. Once waveform data has been loaded into the function generator, Arm it. Shortly thereafter (once the hardware is ready), the Armed property will transition to true, indicating the hardware is ready to react. Once StartTag is received, Armed changes to false and Started is emitted. getArmed() method can be used to obtain the current state.
- sigc::signal< void, bool > SigRunning

  The function generator is currently producing a waveform.
- sigc::signal< void, saftlib::Time > SigStarted

  Function generation has begun.
- sigc::signal< void, saftlib::Time, bool, bool, bool > SigStopped

  Function generation has ended.
- sigc::signal< void > Refill

  More tuples must be appended to ensure uninterrupted waveform.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.57.1 Detailed Description

Function Generator for creating timing triggered waveforms.

The function generator creates a waveform from a piecewise continuous sequence of second order polynomials. These polynomials are specified using a fixed-width ¨parameter tuple¨ format. Once triggered, the function generator outputs samples created from each tupled polynomial until it exhausts the specified piecewise width of the polynomial, whereupon it begins outputting samples from the next tupled polynomial.

The general work-flow to use the FunctionGenerator is to append sets of tuples describing the waveform using the AppendParameterSet function. Then, select a SCUbus timing tag (StartTag) whose appearance should trigger waveform generation. Finally, the FunctionGenerator is armed by calling Arm, whereafter it may be triggered manually via the SCUbusActionSink->InjectTag method, or by a timing event sent by the data master, which is configured to create the StartTag.

If you need an unending waveform, follow the steps above, but additionally monitor the Refill signal, and append additional parameter tuples until low_fill is false.

The sequence of signals of successful waveform generation occur in this order: Enabled(true) Armed(true) Armed(false) Started Running(true) Refill Running(false) Stopped Enabled(false)

7.57.2 Member Function Documentation

7.57.2.1 Abort()

```
void saftlib::FunctionGenerator_Proxy::Abort ( )
```

Abort waveform generation.

This directs the hardware to stop waveform generation. If the function generator was Armed, it is disarmed and disabled, without outputting any waveform data. If the function generator is running, output is Stopped at the current value and disabled. Aborting a function generator takes time, so even after a call to Abort, the function generator might still be Started. However, it will reach the disabled state as quickly as it can, transitioning through Stopped as usual. If the Owner of a FunctionGenerator quits without running Disown, the Abort is run automatically.

7.57.2.2 AppendParameterSet()

```
bool saftlib::FunctionGenerator_Proxy::AppendParameterSet (
            const std::vector< int16_t > & coeff_a,
            const std::vector< int16_t > & coeff_b,
            const std::vector< int32_t > & coeff_c,
            const std::vector< unsigned char > & step,
            const std::vector< unsigned char > & freq,
            const std::vector< unsigned char > & shift_a,
            const std::vector< unsigned char > & shift_b )
```

Append parameter tuples describing waveform to generate.

Parameters

coeff_a	Quadratic coefficient (a), 16-bit signed
coeff_b	Linear coefficient (b), 16-bit signed
coeff_c	Constant coefficient (c), 32-bit signed
step	Number of interpolated samples (0=250, 1=500, 2=1000, 3=2000, 4=4000, 5=8000, 6=16000, or 7=32000)
freq	Output sample frequency (0=16kHz, 1=32kHz, 2=64kHz, 3=125kHz, 4=250kHz, 5=500kHz, 6=1MHz, or 7=2MHz)
shift_a	Exponent of coeff_a, 6-bit unsigned; a = coeff_a∗2^shift_a
shift_b	Exponent of coeff_b, 6-bit unsigned; b = coeff_b∗2^shift_b
low_fill	Fill level remains too low

This function appends the parameter vectors (which must be equal in length) to the FIFO of remaining waveform to generate. Each parameter set (coefficients) describes a number of output samples in the generated wave form. Parameter sets are executed in order until no more remain.

If the fill level is not high enough, this method returns true. Only once this function has returned false can you await the Refill signal.

At each step, the function generator outputs high_bits(c∗2^32 + b∗t + c∗t∗t), where t ranges from 0 to numSteps-1. high_bits are the high OutputWindowSize bits of the resulting 64-bit signed value.

7.57.2.3 Arm()

```
void saftlib::FunctionGenerator_Proxy::Arm ( )
```

Enable the function generator and arm it.

A function generator can only be Armed if FillLevel is non-zero. An Enabled function generator can not be Armed again until it either completes waveform generation or the user calls Abort. Arming a function generator takes time. Wait for Armed to transition to true before sending StartTag.

7.57.2.4 Flush()

```
void saftlib::FunctionGenerator_Proxy::Flush ( )
```

Empty the parameter tuple set.

Flush may only be called when not Enabled. Flush does not clear the ExecutedParameterCount.

7.57.2.5 getArmed()

```
bool saftlib::FunctionGenerator_Proxy::getArmed ( )
```

Upon receipt of StartTag, output will begin.

If a function generator is Armed, it is also Enabled. Once waveform data has been loaded into the function generator, Arm it. Shortly thereafter (once the hardware is ready), the Armed property will transition to true, indicating the hardware is ready to react. Once StartTag is received, Armed changes to false and Started is emitted. SigArmed is emitted when this changes.

Returns

true if armed

7.57.2.6 getDeviceNumber()

```
unsigned char saftlib::FunctionGenerator_Proxy::getDeviceNumber ( )
```

number of the hardware macro inside of the slave card

Returns

number of the hardware macro inside of the slave card

7.57.2.7 getEnabled()

```
bool saftlib::FunctionGenerator_Proxy::getEnabled ( )
```

Hardware is allowed to generate an output waveform. If a function generator is disabled, its output is constant. Enabled is set to true using Arm and transitions to false either upon completion or after a call to Abort. SigEnabled is emitted when this changes.

Returns

true if enabled

7.57.2.8 getOutputWindowSize()

```
unsigned char saftlib::FunctionGenerator_Proxy::getOutputWindowSize ( )
```

Number of bits output by the function generator.

Returns

Number of bits output by the function generator

7.57.2.9 getRunning()

```
bool saftlib::FunctionGenerator_Proxy::getRunning ( )
```

The function generator is currently producing a waveform.

Function generation is started by sending a SCUbus tag which matches the StartTag property of an Armed function generator. When this property transitions to true, the Started signal is emitted. When this property transitions to false, the Stopped signal is emitted. SigRunning or is emitted when this changes.

Returns

true if running

7.57.2.10 getSCUbusSlot()

```
unsigned char saftlib::FunctionGenerator_Proxy::getSCUbusSlot ( )
```

Slot number of the slave card.

Returns

Slot number of the slave card

7.57.2.11 getStartTag()

```
uint32_t saftlib::FunctionGenerator_Proxy::getStartTag ( )
```

The SCUbus tag which causes function generation to begin.

If the function generator is Armed and this tag is sent to the SCUbus, then the function generator will begin generating the output waveform. StartTag may only be set when the FunctionGenerator is not Enabled.

Returns

SCUbus tag

7.57.2.12 getVersion()

```
uint32_t saftlib::FunctionGenerator_Proxy::getVersion ( )
```

Version of the hardware macro.

Returns

Version of the hardware macro

7.57.2.13 ReadExecutedParameterCount()

```
uint32_t saftlib::FunctionGenerator_Proxy::ReadExecutedParameterCount ( )
```

Number of parameter tuples executed by hardware.

This counts the total number of parameter tuples executed since the last Started signal. Obviously, if the function generator is running, the returned value will be old.

Returns

Number tuples executed by hardware.

7.57.2.14 ReadFillLevel()

```
uint64_t saftlib::FunctionGenerator_Proxy::ReadFillLevel ( )
```

Remaining waveform data in nanoseconds.

The SAFTd has sufficient parameters buffered to supply the function generator with data for the specified time in nanoseconds. Note, due to the slow nature of software, if the function generator is currently running, the read value will already be out-of-date upon return. This property should be used for informational use only.

Returns

Remaining waveform data in nanoseconds.

7.57.2.15 signal_dispatch()

```
bool saftlib::FunctionGenerator_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

7.57.3 Member Data Documentation

7.57.3.1 Refill

```
sigc::signal<void> saftlib::FunctionGenerator_Proxy::Refill
```

More tuples must be appended to ensure uninterrupted waveform.

In order to guarantee an uninterrupted supply of data to the function generator, there should be data buffered in the SAFTd. When the buffer fill level gets too low, this signal is emitted, and you should run AppendParameterSet. If you do not, function generation will cease, signalling successful completion of the waveform with Stopped.

7.57.3.2 SigRunning

```
sigc::signal<void, bool> saftlib::FunctionGenerator_Proxy::SigRunning
```

The function generator is currently producing a waveform.

Function generation is started by sending a SCUbus tag which matches the StartTag property of an Armed function generator. When this property transitions to true, the Started signal is emitted. When this property transitions to false, the Stopped signal is emitted. getRunning() method can be used to obtain the current state.

7.57.3.3 SigStarted

```
sigc::signal<void, saftlib::Time> saftlib::FunctionGenerator_Proxy::SigStarted
```

Function generation has begun.

This signal notifies software when function generation has begun, possibly to update a GUI or other user-facing status information.

Parameters

time	Time when function generation began in nanoseconds since 1970

7.57.3.4 SigStopped

```
sigc::signal<void, saftlib::Time, bool, bool, bool> saftlib::FunctionGenerator_Proxy::Sig↩
Stopped
```

Function generation has ended.

Parameters

time	Time when function generation ended in nanoseconds since 1970
abort	stopped due to a call to Abort
hardwareMacroUnderflow	A fatal error, indicating the SCUbus is congested
microControllerUnderflow	A fatal error, indicating the host CPU is overloaded

The function generator stops either successfully (when all data has been sent), or it stops due to an error. When an error occurs, the function generator stops and holds its most recent value. This can occur due to two causes:

hardwareMacroUnderflow, a fatal error indicating the hardware ran out of data. If the SCUbus is too busy, it can happen that the waveform data stored in the function generator HDL is not refilled in time. This error can only be mitigated by ensuring that the function generator does not share the SCUbus with other users.

microControllerUnderflow, a fatal error indicating the microcontroller ran out of data. If the host CPU running this software is too busy, it can happen that the waveform data is not delivered to the microcontroller before the microcontroller runs out of data. This error can be mitigated by reducing the number of busy processes running on the system.

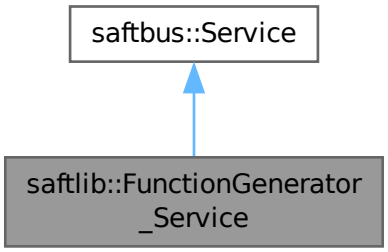
Once the function generator has stopped, ExecutedParameterCount remains valid until the next time the function generator starts. After stopping, regardless of if the generation was successful or not, the parameter FIFO is cleared, Enabled is false, and this signal emitted.

The documentation for this class was generated from the following files:

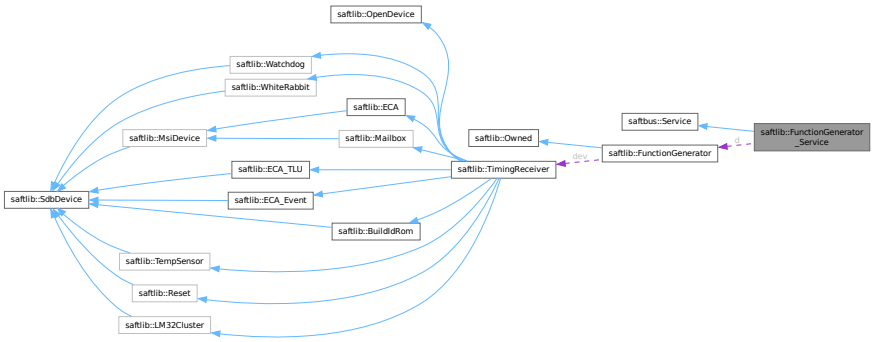
- src/FunctionGenerator_Proxy.hpp
- src/FunctionGenerator_Proxy.cpp

7.58 saftlib::FunctionGenerator_Service Class Reference

Inheritance diagram for saftlib::FunctionGenerator_Service:



Collaboration diagram for saftlib::FunctionGenerator_Service:



Public Types

- typedef FunctionGenerator **DriverType**

Public Member Functions

- **FunctionGenerator_Service** (FunctionGenerator *instance, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **SigEnabled_dispatch_function** (bool arg_1)
- void **SigArmed_dispatch_function** (bool arg_1)
- void **SigRunning_dispatch_function** (bool arg_1)
- void **SigStarted_dispatch_function** (saftlib::Time arg_1)
- void **SigStopped_dispatch_function** (saftlib::Time arg_1, bool arg_2, bool arg_3, bool arg_4)
- void **Refill_dispatch_function** ()
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- FunctionGenerator * **d**
- sigc::connection **SigEnabled_connection**
- sigc::connection **SigArmed_connection**
- sigc::connection **SigRunning_connection**
- sigc::connection **SigStarted_connection**
- sigc::connection **SigStopped_connection**
- sigc::connection **Refill_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.58.1 Member Function Documentation

7.58.1.1 call()

```
void saftlib::FunctionGenerator_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

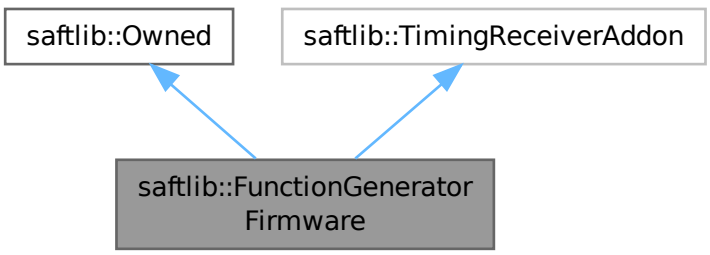
- src/FunctionGenerator_Service.hpp
- src/FunctionGenerator_Service.cpp

7.59 saftlib::FunctionGeneratorFirmware Class Reference

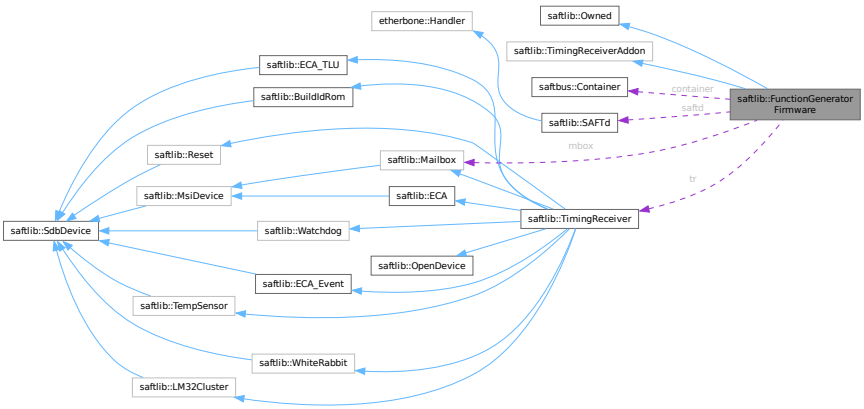
Representation of the FunctionGenerator Firmware.

`#include <FunctionGeneratorFirmware.hpp>`

Inheritance diagram for saftlib::FunctionGeneratorFirmware:



Collaboration diagram for saftlib::FunctionGeneratorFirmware:



Public Member Functions

- std::map< std::string, std::map< std::string, std::string > > getObjects ()
- std::string **getObjectPath** ()
- **FunctionGeneratorFirmware** (saftbus::Container *container, SAFTd *saft_daemon, TimingReceiver *timing_receiver)
- uint32_t **getVersion** () const
- std::map< std::string, std::string > Scan ()

  Scan bus for fg channels.
- std::map< std::string, std::string > **ScanMasterFg** ()
- std::map< std::string, std::string > **ScanFgChannels** ()
- void **removeMasterFunctionGenerator** ()

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

- virtual std::map< std::string, std::map< std::string, std::string > > **getObjects** ()=0

Protected Member Functions

- void **clear** ()

  remove all owned FunctionGenerators or MasterFunctionGenerator
- void **firmware_rescan** (int host_to_lm32_mailbox_slot_idx)
- bool **nothing_runs** ()

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes

- saftbus::Container * **container**
- std::string **objectPath**
- SAFTd * **saftd**
- TimingReceiver * **tr**
- Mailbox * **mbox**
- etherbone::Device & **device**
- std::map< std::string, std::shared_ptr< FunctionGenerator > > **fgs**
- std::shared_ptr< MasterFunctionGenerator > **mfg**
- bool **have_fg_firmware**
- eb_data_t **magic**
- eb_data_t **version**
- eb_address_t **fgb**
- std::map< std::string, std::map< std::string, std::string > > **addon_objects**

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

7.59.1 Detailed Description

Representation of the FunctionGenerator Firmware.

It can trigger the firmware to do a channel rescan. The channels can be assigned to individual FunctionGenerator objects or to a single MasterFunctionGenerator object

7.59.2 Member Function Documentation

7.59.2.1 getObjects()

```
std::map< std::string, std::map< std::string, std::string > > saftlib::FunctionGenerator↩
Firmware::getObjects ( )  [virtual]
```

Implements saftlib::TimingReceiverAddon.

7.59.2.2 Scan()

```
std::map< std::string, std::string > saftlib::FunctionGeneratorFirmware::Scan ( )
```

Scan bus for fg channels.

This function should only be called if no fg-channel is active. If any channel is active and this function is called a saftbus::Error will be thrown!

Returns

fgs found.

The documentation for this class was generated from the following files:

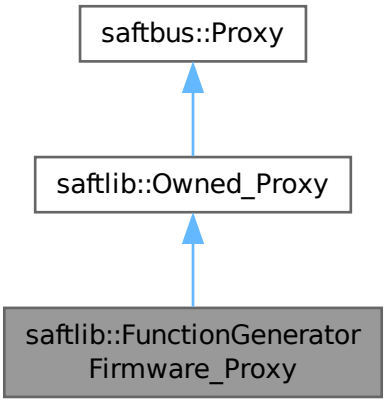
- src/FunctionGeneratorFirmware.hpp
- src/FunctionGeneratorFirmware.cpp

7.60 saftlib::FunctionGeneratorFirmware_Proxy Class Reference

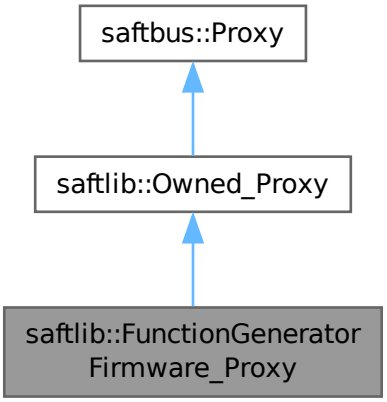
Representation of the FunctionGenerator Firmware.

```
#include <FunctionGeneratorFirmware_Proxy.hpp>
```

Inheritance diagram for saftlib::FunctionGeneratorFirmware_Proxy:



Collaboration diagram for saftlib::FunctionGeneratorFirmware_Proxy:



Public Member Functions

- **FunctionGeneratorFirmware_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal↩ _group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_↩ interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::map< std::string, std::string > Scan ()

  Scan bus for fg channels.
- std::map< std::string, std::string > **ScanMasterFg** ()
- std::map< std::string, std::string > **ScanFgChannels** ()

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< FunctionGeneratorFirmware_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.60.1 Detailed Description

Representation of the FunctionGenerator Firmware.

It can trigger the firmware to do a channel rescan. The channels can be assigned to individual FunctionGenerator objects or to a single MasterFunctionGenerator object

7.60.2 Member Function Documentation

7.60.2.1 Scan()

```
std::map< std::string, std::string > saftlib::FunctionGeneratorFirmware_Proxy::Scan ( )
```

Scan bus for fg channels.

This function should only be called if no fg-channel is active. If any channel is active and this function is called a saftbus::Error will be thrown!

Returns

fgs found.

7.60.2.2 signal_dispatch()

```
bool saftlib::FunctionGeneratorFirmware_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

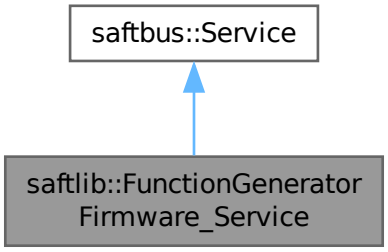
Implements saftbus::Proxy.

The documentation for this class was generated from the following files:

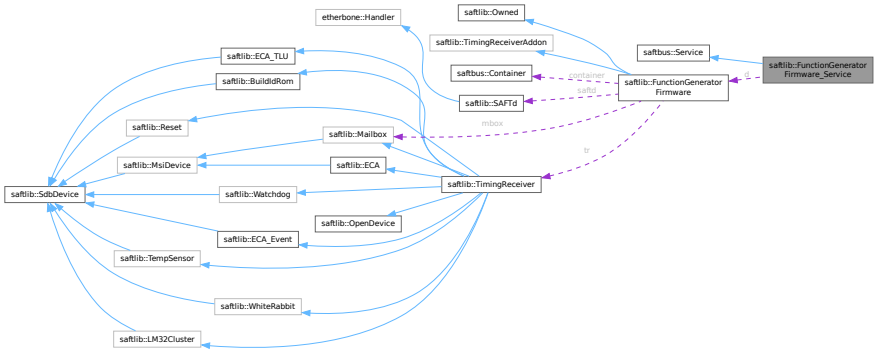
- src/FunctionGeneratorFirmware_Proxy.hpp
- src/FunctionGeneratorFirmware_Proxy.cpp

7.61 saftlib::FunctionGeneratorFirmware_Service Class Reference

Inheritance diagram for saftlib::FunctionGeneratorFirmware_Service:



Collaboration diagram for saftlib::FunctionGeneratorFirmware_Service:



Public Types

- typedef FunctionGeneratorFirmware **DriverType**

Public Member Functions

- **FunctionGeneratorFirmware_Service** (FunctionGeneratorFirmware *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- FunctionGeneratorFirmware * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.61.1 Member Function Documentation

7.61.1.1 call()

```
void saftlib::FunctionGeneratorFirmware_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

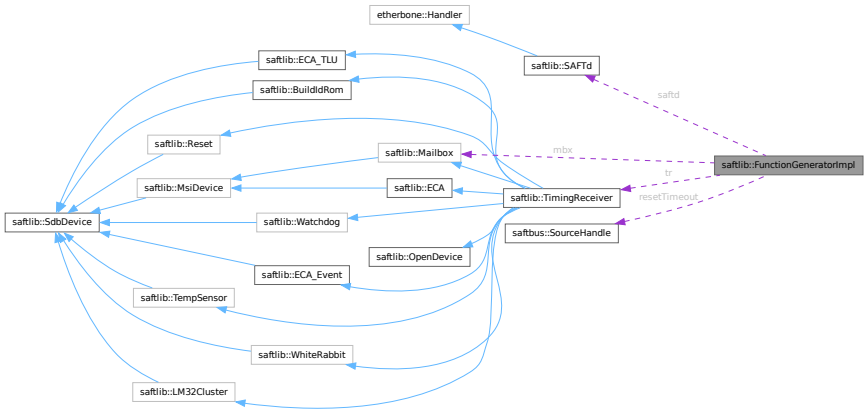
Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/FunctionGeneratorFirmware_Service.hpp
- src/FunctionGeneratorFirmware_Service.cpp

7.62 saftlib::FunctionGeneratorImpl Class Reference

Collaboration diagram for saftlib::FunctionGeneratorImpl:



Public Member Functions

- **FunctionGeneratorImpl** (SAFTd ∗saft_daemon, TimingReceiver ∗timing_receiver, std::shared_ptr< std↩ ::vector< int > > channel_allocation, eb_address_t fgb, int mbox_slot, unsigned n_channels, unsigned buf↩ _size, unsigned idx, uint32_t macro)
- template<typename Iter >
  bool **appendParameterTuples** (Iter it, Iter end)
- bool **appendParameterTuples** (std::vector< ParameterTuple > parameters)
- void **Arm** ()
- void **Abort** ()
- uint64_t **ReadFillLevel** ()

- bool **appendParameterSet** (const std::vector< int16_t > &coeff_a, const std::vector< int16_t > &coeff_b, const std::vector< int32_t > &coeff_c, const std::vector< unsigned char > &step, const std::vector< unsigned char > &freq, const std::vector< unsigned char > &shift_a, const std::vector< unsigned char > &shift_b)
- void **Flush** ()
- uint32_t **getVersion** () const
- unsigned char **getSCUbusSlot** () const
- unsigned char **getDeviceNumber** () const
- unsigned char **getOutputWindowSize** () const
- bool **getEnabled** () const
- bool **getArmed** () const
- bool **getRunning** () const
- uint32_t **getStartTag** () const
- uint32_t **ReadExecutedParameterCount** ()
- void **setStartTag** (uint32_t val)
- std::string **GetName** ()
- void **flush** ()
- void **arm** ()
- void **Reset** ()

Public Attributes

- sigc::signal< void, bool > **signal_enabled**
- sigc::signal< void, bool > **signal_running**
- sigc::signal< void, bool > **signal_armed**
- sigc::signal< void > **signal_refill**
- sigc::signal< void, uint64_t > **signal_started**
- sigc::signal< void, uint64_t, bool, bool, bool > **signal_stopped**

Protected Member Functions

- bool **lowFill** () const
- void **irq_handler** (eb_data_t msi)
- void **refill** (bool)
- void **releaseChannel** ()
- void **acquireChannel** ()
- bool **ResetFailed** ()
- void **ownerQuit** ()
- void **fifo_push_back** (const ParameterTuple &tuple)

Protected Attributes

- SAFTd * **saftd**
- TimingReceiver * **tr**
- Mailbox * **mbx**
- etherbone::Device & **device**
- std::shared_ptr< std::vector< int > > **allocation**
- eb_address_t **shm**
- int **mb_slot**
- unsigned **num_channels**
- unsigned **buffer_size**
- unsigned int **index**

- unsigned char **scubusSlot**
- unsigned char **deviceNumber**
- unsigned char **version**
- unsigned char **outputWindowSize**
- std::unique_ptr< saftlib::IRQ > **irq**
- std::unique_ptr< Mailbox::Slot > **host_slot**
- int **channel**
- bool **enabled**
- bool **armed**
- bool **running**
- bool **abort**
- saftbus::SourceHandle **resetTimeout**
- uint32_t **startTag**
- unsigned **executedParameterCount**
- unsigned **mbx_slot**
- eb_address_t **mailbox_slot_address**
- uint64_t **fillLevel**
- unsigned **filled**
- boost::circular_buffer< ParameterTuple > **fifo**
- unsigned **fg_fifo_max_size**

Friends

- class **MasterFunctionGenerator**

The documentation for this class was generated from the following files:

- src/FunctionGeneratorImpl.hpp
- src/FunctionGeneratorImpl.cpp

7.63 saftbus::ClientConnection::Impl Struct Reference

Public Attributes

- struct pollfd **pfd**
- int **client_id**
- std::mutex **fd_mutex**
- std::mutex **connection_mutex**

Static Public Attributes

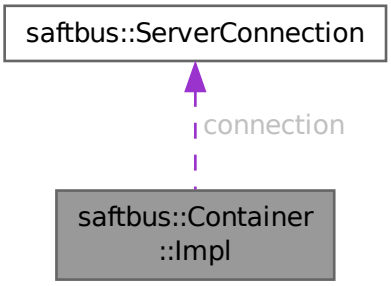
- static std::mutex **base_socket_mutex**

The documentation for this struct was generated from the following file:

- saftbus/client.cpp

7.64 saftbus::Container::Impl Struct Reference

Collaboration diagram for saftbus::Container::Impl:



Public Member Functions

- unsigned **generate_saftbus_object_id** ()
- void **reset_children_first** (const std::string &object_path)
- void **clear** ()

Public Attributes

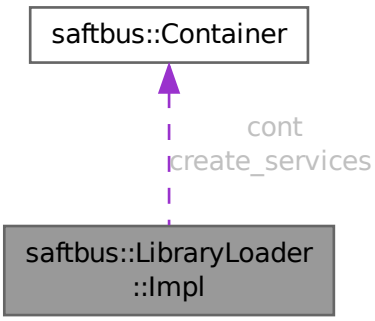
- std::vector< std::pair< std::string, std::unique_ptr< LibraryLoader > > > **plugins**
- ServerConnection * **connection**
- std::map< unsigned, std::unique_ptr< Service > > **objects**
- std::map< std::string, unsigned > **object_path_lookup_table**
- std::vector< Service * > **removed_services**
- std::map< std::string, std::function< std::string(void)> > **additional_info_callbacks**

The documentation for this struct was generated from the following file:

- saftbus/service.cpp

7.65 saftbus::LibraryLoader::Impl Struct Reference

Collaboration diagram for saftbus::LibraryLoader::Impl:



Public Attributes

- void * **handle**
- saftbus::Container * **cont**
- create_services_function **create_services**

The documentation for this struct was generated from the following file:

- saftbus/plugins.cpp

7.66 saftbus::Loop::Impl Struct Reference

Public Attributes

- std::vector< std::unique_ptr< Source > > **added_sources**
- std::vector< std::unique_ptr< Source > > **sources**
- bool **running**
- int **running_depth**
- long **id**

Static Public Attributes

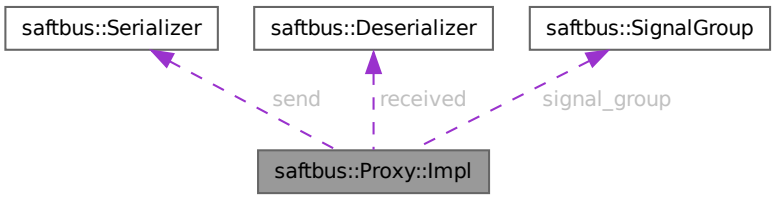
- static long **id_counter** = 0

The documentation for this struct was generated from the following file:

- saftbus/loop.cpp

7.67 saftbus::Proxy::Impl Struct Reference

Collaboration diagram for saftbus::Proxy::Impl:



Public Attributes

- std::mutex **proxy_mutex**
- int **saftbus_object_id**
- int **client_id**
- int **signal_group_id**
- Serializer **send**
- Deserializer **received**
- SignalGroup * **signal_group**
- std::vector< std::string > **interface_names**
- std::map< std::string, int > **interface_name2no_map**

Static Public Attributes

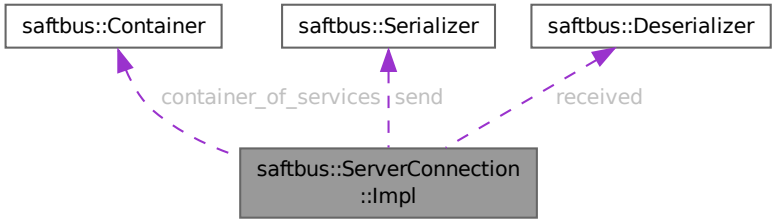
- static std::shared_ptr< ClientConnection > **connection**
- static std::mutex **connection_mutex**

The documentation for this struct was generated from the following file:

- saftbus/client.cpp

7.68 saftbus::ServerConnection::Impl Struct Reference

Collaboration diagram for saftbus::ServerConnection::Impl:



Public Member Functions

- **Impl** (ServerConnection *connection)
- bool **accept_client** (int fd, int condition)
- bool **handle_client_request** (int fd, int condition)
- void **client_hung_up** (int client_fd)

Public Attributes

- Container **container_of_services**
- std::vector< std::unique_ptr< Client > > **clients**
- Serializer **send**
- Deserializer **received**
- int **calling_client_id**

The documentation for this struct was generated from the following file:

- saftbus/server.cpp

7.69 saftbus::Service::Impl Struct Reference

Public Member Functions

- void **remove_signal_fd** (int fd)

Public Attributes

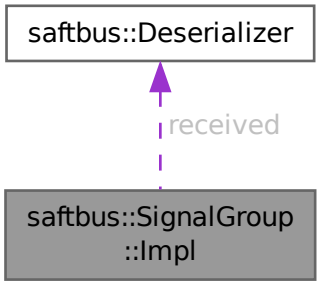
- int **owner**
- std::map< int, int > **signal_fds_use_count**
- std::vector< std::string > **interface_names**
- std::string **object_path**
- uint64_t **object_id**
- std::function< void()> **destruction_callback**
- bool **destroy_if_owner_quits**

The documentation for this struct was generated from the following file:

- saftbus/service.cpp

7.70 saftbus::SignalGroup::Impl Struct Reference

Collaboration diagram for saftbus::SignalGroup::Impl:



Public Attributes

- struct pollfd **pfd**
- int **fd_pair** [2]
- int **signal_group_id**
- Deserializer **received**
- std::vector< Proxy * > **proxies**
- std::mutex **signal_group_mutex**
- std::mutex **fd_mutex**

The documentation for this struct was generated from the following file:

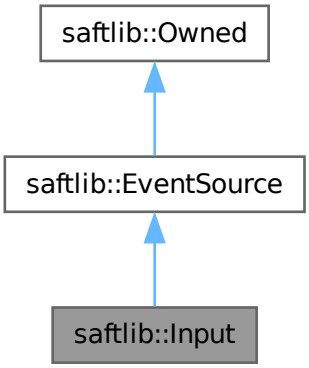
- saftbus/client.cpp

7.71 saftlib::Input Class Reference

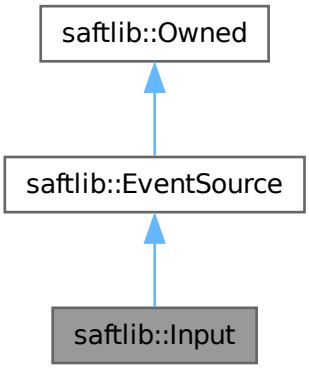
Interface to a TimingReceiver input (such as button or Lemo-IO)

`#include <Input.hpp>`

Inheritance diagram for saftlib::Input:



Collaboration diagram for saftlib::Input:



Public Member Functions

- **Input** (ECA_TLU &eca_tlu, const std::string &input_object_path, const std::string &output_partner_path, unsigned eca_tlu_chan, Io *io, saftbus::Container *container=nullptr)
- bool ReadInput ()

  Read the current input value.
- uint32_t getStableTime () const

  Deglitch threshold for the input.
- void setStableTime (uint32_t val)

  Deglitch threshold for the input.
- uint32_t getIndexIn () const

Each IO on hardware has a unique index.

- bool getSpecialPurposeIn () const

  Is the special function enabled.
- void **setSpecialPurposeIn** (bool val)

  enable or disable special function
- bool getSpecialPurposeInAvailable () const

  Some inputs have special purpose configuration. What exacly depends on the input.
- bool getGateIn () const

  Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)
- void setGateIn (bool val)

  Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)
- bool getInputTermination () const

  50 ohm input termination
- void setInputTermination (bool val)

  enable or disable 50 ohm input termination
- bool getInputTerminationAvailable () const

  Inputs can have a configurable 50 ohm termination.
- std::string getLogicLevelIn () const

  Inputs support different logic levels (LVDS, LVTTL, ...)
- std::string getTypeIn () const

  IO type (GPIO, LVDS, ...)
- std::string getOutput () const

  path of the Output object for the same physical IO
- uint64_t getResolution () const

  The precision of generated timestamps in nanoseconds.
- uint32_t getEventBits () const

  How many bits of external data are included in the ID.
- bool getEventEnable () const

  Should the event source generate events.
- uint64_t getEventPrefix () const

  Combined with low EventBits to create generated IDs.
- void setEventEnable (bool val)
- void setEventPrefix (uint64_t val)

Public Member Functions inherited from saftlib::EventSource

- **EventSource** (const std::string &object_path, const std::string &name, saftbus::Container *container)
- virtual uint64_t getResolution () const =0

  The precision of generated timestamps in nanoseconds.
- virtual uint32_t getEventBits () const =0

  How many bits of external data are included in the ID.
- virtual bool getEventEnable () const =0

  Should the event source generate events.
- virtual void **setEventEnable** (bool val)=0
- virtual uint64_t getEventPrefix () const =0

  Combined with low EventBits to create generated IDs.
- virtual void **setEventPrefix** (uint64_t val)=0
- std::string **getObjectPath** () const
- std::string **getObjectName** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

7.71.1 Detailed Description

Interface to a TimingReceiver input (such as button or Lemo-IO)

7.71.2 Member Function Documentation

7.71.2.1 getEventBits()

```
uint32_t saftlib::Input::getEventBits ( ) const  [virtual]
```

How many bits of external data are included in the ID.

Returns

How many bits of external data are included in the ID

Implements saftlib::EventSource.

7.71.2.2 getEventEnable()

```
bool saftlib::Input::getEventEnable ( ) const  [virtual]
```

Should the event source generate events.

Returns

true if the event source generates events

Implements saftlib::EventSource.

7.71.2.3 getEventPrefix()

```
uint64_t saftlib::Input::getEventPrefix ( ) const  [virtual]
```

Combined with low EventBits to create generated IDs.

Returns

the event prefix

Implements saftlib::EventSource.

7.71.2.4 getGateIn()

```
bool saftlib::Input::getGateIn ( ) const
```

Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)

Returns

true if gate is open

7.71.2.5 getIndexIn()

```
uint32_t saftlib::Input::getIndexIn ( ) const
```

Each IO on hardware has a unique index.

Returns

IO index

7.71.2.6 getInputTermination()

```
bool saftlib::Input::getInputTermination ( ) const
```

50 ohm input termination

Returns

true if termination is enabled, false otherwise.

Some inputs need termination to receive a clean input signal. However, if the same IO is used as an Output, termination should probably be disabled. This defaults to on. See also OutputEnable if this is an inoutput.

7.71.2.7 getInputTerminationAvailable()

```
bool saftlib::Input::getInputTerminationAvailable ( ) const
```

Inputs can have a configurable 50 ohm termination.

Returns

true if (input) termination be configured. false otherwise.

7.71.2.8 getLogicLevelIn()

```
std::string saftlib::Input::getLogicLevelIn ( ) const
```

Inputs support different logic levels (LVDS, LVTTL, ...)

Returns

Logic level of the input (LVDS, LVTTL, ...)

7.71.2.9 getOutput()

```
std::string saftlib::Input::getOutput ( ) const
```

path of the Output object for the same physical IO

Returns

Empty string if this is not an output. Path of the Output object for the same physical IO otherwise.

7.71.2.10 getResolution()

```
uint64_t saftlib::Input::getResolution ( ) const  [virtual]
```

The precision of generated timestamps in nanoseconds.

Returns

The precision of generated timestamps in nanoseconds.

Implements saftlib::EventSource.

7.71.2.11 getSpecialPurposeIn()

```
bool saftlib::Input::getSpecialPurposeIn ( ) const
```

Is the special function enabled.

Returns

true if special function is enabled, false otherwise.

7.71.2.12 getSpecialPurposeInAvailable()

```
bool saftlib::Input::getSpecialPurposeInAvailable ( ) const
```

Some inputs have special purpose configuration. What exacly depends on the input.

Returns

true if input has special configuration.

7.71.2.13 getStableTime()

```
uint32_t saftlib::Input::getStableTime ( ) const
```

Deglitch threshold for the input.

Returns

number of nanoseconds input signal needs to be stable before an edge is detected

The number of nanoseconds a signal must remain high or low in order to be considered a valid transition. Increasing this value will not impact the resulting timestamps, but will hide transitions smaller than the threshold. For example, if StableTime=400, then a 5MHz signal would be completely ignored.

7.71.2.14 getTypeIn()

```
std::string saftlib::Input::getTypeIn ( ) const
```

IO type (GPIO, LVDS, ...)

Returns

the IO type

7.71.2.15 ReadInput()

```
bool saftlib::Input::ReadInput ( )
```

Read the current input value.

Returns

The current logic level on the input.

For inoutputs, this may differ from the Output value, if OutputEnable is false. To receive a signal on Input changes, use the EventSource interface to create timing events and monitor these via a SoftwareActionSink.

7.71.2.16 setEventEnable()

```
void saftlib::Input::setEventEnable (
            bool val )  [virtual]
```

Implements saftlib::EventSource.

7.71.2.17 setEventPrefix()

```
void saftlib::Input::setEventPrefix (
            uint64_t val )  [virtual]
```

Implements saftlib::EventSource.

7.71.2.18 setGateIn()

```
void saftlib::Input::setGateIn (
            bool val )
```

Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)

Parameters

val	true enables the gate, false disables it

7.71.2.19 setInputTermination()

```
void saftlib::Input::setInputTermination (
            bool val )
```

enable or disable 50 ohm input termination

Parameters

val	true enables input termination

7.71.2.20 setStableTime()

```
void saftlib::Input::setStableTime (
            uint32_t val )
```

Deglitch threshold for the input.

Parameters

val	number of nanoseconds the input needs to be stable after an edge

The documentation for this class was generated from the following files:

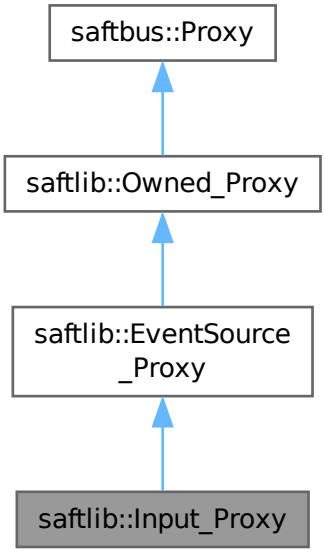
- src/Input.hpp
- src/Input.cpp

7.72 saftlib::Input_Proxy Class Reference

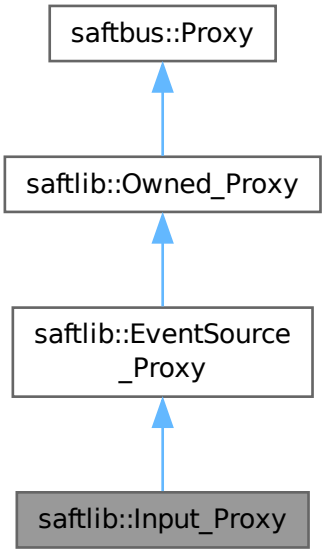
Interface to a TimingReceiver input (such as button or Lemo-IO)

```
#include <Input_Proxy.hpp>
```

Inheritance diagram for saftlib::Input_Proxy:



Collaboration diagram for saftlib::Input_Proxy:



Public Member Functions

- **Input_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩

::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())

- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- bool ReadInput ()

  Read the current input value.
- uint32_t getStableTime ()

  Deglitch threshold for the input.
- void setStableTime (uint32_t val)

  Deglitch threshold for the input.
- uint32_t getIndexIn ()

  Each IO on hardware has a unique index.
- bool getSpecialPurposeIn ()

  Is the special function enabled.
- void **setSpecialPurposeIn** (bool val)

  enable or disable special function
- bool getSpecialPurposeInAvailable ()

  Some inputs have special purpose configuration. What exacly depends on the input.
- bool getGateIn ()

  Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)
- void setGateIn (bool val)

  Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)
- bool getInputTermination ()

  50 ohm input termination
- void setInputTermination (bool val)

  enable or disable 50 ohm input termination
- bool getInputTerminationAvailable ()

  Inputs can have a configurable 50 ohm termination.
- std::string getLogicLevelIn ()

  Inputs support different logic levels (LVDS, LVTTL, ...)
- std::string getTypeIn ()

  IO type (GPIO, LVDS, ...)
- std::string getOutput ()

  path of the Output object for the same physical IO

Public Member Functions inherited from saftlib::EventSource_Proxy

- **EventSource_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- virtual uint64_t getResolution ()

  The precision of generated timestamps in nanoseconds.
- virtual uint32_t getEventBits ()

  How many bits of external data are included in the ID.
- virtual bool getEventEnable ()

  Should the event source generate events.
- virtual void **setEventEnable** (bool val)
- virtual uint64_t getEventPrefix ()

  Combined with low EventBits to create generated IDs.
- virtual void **setEventPrefix** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< Input_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal↩ _group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::EventSource_Proxy

- static std::shared_ptr< EventSource_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.72.1 Detailed Description

Interface to a TimingReceiver input (such as button or Lemo-IO)

7.72.2 Member Function Documentation

7.72.2.1 getGateIn()

```
bool saftlib::Input_Proxy::getGateIn ( )
```

Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)

Returns

true if gate is open

7.72.2.2 getIndexIn()

```
uint32_t saftlib::Input_Proxy::getIndexIn ( )
```

Each IO on hardware has a unique index.

Returns

IO index

7.72.2.3 getInputTermination()

```
bool saftlib::Input_Proxy::getInputTermination ( )
```

50 ohm input termination

Returns

true if termination is enabled, false otherwise.

Some inputs need termination to receive a clean input signal. However, if the same IO is used as an Output, termination should probably be disabled. This defaults to on. See also OutputEnable if this is an inoutput.

7.72.2.4 getInputTerminationAvailable()

```
bool saftlib::Input_Proxy::getInputTerminationAvailable ( )
```

Inputs can have a configurable 50 ohm termination.

Returns

true if (input) termination be configured. false otherwise.

7.72.2.5 getLogicLevelIn()

```
std::string saftlib::Input_Proxy::getLogicLevelIn ( )
```

Inputs support different logic levels (LVDS, LVTTL, ...)

Returns

Logic level of the input (LVDS, LVTTL, ...)

7.72.2.6 getOutput()

```
std::string saftlib::Input_Proxy::getOutput ( )
```

path of the Output object for the same physical IO

Returns

Empty string if this is not an output. Path of the Output object for the same physical IO otherwise.

7.72.2.7 getSpecialPurposeIn()

```
bool saftlib::Input_Proxy::getSpecialPurposeIn ( )
```

Is the special function enabled.

Returns

true if special function is enabled, false otherwise.

7.72.2.8 getSpecialPurposeInAvailable()

```
bool saftlib::Input_Proxy::getSpecialPurposeInAvailable ( )
```

Some inputs have special purpose configuration. What exacly depends on the input.

Returns

true if input has special configuration.

7.72.2.9 getStableTime()

```
uint32_t saftlib::Input_Proxy::getStableTime ( )
```

Deglitch threshold for the input.

Returns

number of nanoseconds input signal needs to be stable before an edge is detected

The number of nanoseconds a signal must remain high or low in order to be considered a valid transition. Increasing this value will not impact the resulting timestamps, but will hide transitions smaller than the threshold. For example, if StableTime=400, then a 5MHz signal would be completely ignored.

7.72.2.10 getTypeIn()

```
std::string saftlib::Input_Proxy::getTypeIn ( )
```

IO type (GPIO, LVDS, ...)

Returns

the IO type

7.72.2.11 ReadInput()

```
bool saftlib::Input_Proxy::ReadInput ( )
```

Read the current input value.

Returns

The current logic level on the input.

For inoutputs, this may differ from the Output value, if OutputEnable is false. To receive a signal on Input changes, use the EventSource interface to create timing events and monitor these via a SoftwareActionSink.

7.72.2.12 setGateIn()

```
void saftlib::Input_Proxy::setGateIn (
            bool val )
```

Inputs have a logic gate to prevent inputs to propagate further into the system (ECA_TLU)

Parameters

val	true enables the gate, false disables it

7.72.2.13 setInputTermination()

```
void saftlib::Input_Proxy::setInputTermination (
            bool val )
```

enable or disable 50 ohm input termination

Parameters

val	true enables input termination

7.72.2.14 setStableTime()

```
void saftlib::Input_Proxy::setStableTime (
            uint32_t val )
```

Deglitch threshold for the input.

Parameters

val	number of nanoseconds the input needs to be stable after an edge

7.72.2.15 signal_dispatch()

```
bool saftlib::Input_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

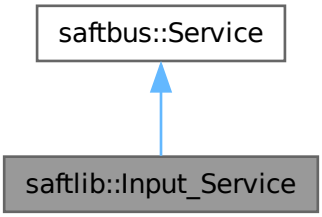
Reimplemented from saftlib::EventSource_Proxy.

The documentation for this class was generated from the following files:

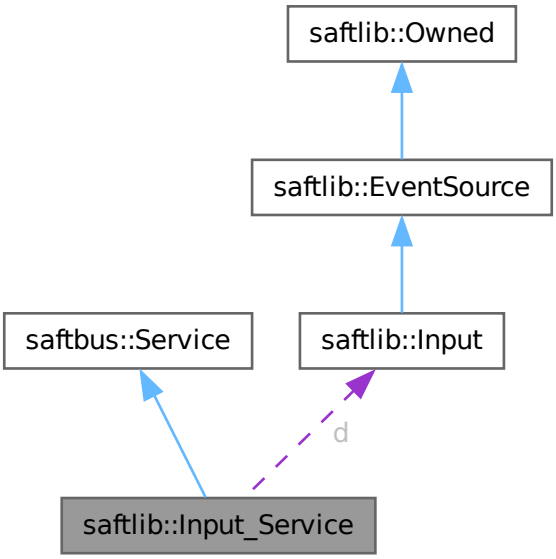
- src/Input_Proxy.hpp
- src/Input_Proxy.cpp

7.73 saftlib::Input_Service Class Reference

Inheritance diagram for saftlib::Input_Service:



Collaboration diagram for saftlib::Input_Service:



Public Types

- typedef Input **DriverType**

Public Member Functions

- **Input_Service** (Input *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- Input * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.73.1 Member Function Documentation

7.73.1.1 call()

```
void saftlib::Input_Service::call (
             unsigned interface_no,
             unsigned function_no,
             int client_fd,
             saftbus::Deserializer & received,
             saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/Input_Service.hpp
- src/Input_Service.cpp

7.74 saftlib::Io Class Reference

representaion of a single IO on a TimingReceiver

`#include <Io.hpp>`

Public Member Functions

- **Io** (etherbone::Device &device, const std::string &io_name, unsigned io_direction, unsigned io_channel, unsigned eca_in, unsigned eca_out, unsigned io_index, unsigned io_special_purpose, unsigned io_logic_level, bool io_oe_available, bool io_term_available, bool io_spec_out_available, bool io_spec_in_available, eb_address_t io_control_addr, SerdesClockGen &clkgen)
- const std::string & **getName** () const

- unsigned **getDirection** () const
- uint32_t **getIndexOut** () const
- uint32_t **getIndexIn** () const
- uint32_t **getEcaIn** () const
- uint32_t **getEcaOut** () const
- void **WriteOutput** (bool value)
- bool **ReadOutput** ()
- bool **ReadCombinedOutput** ()
- bool **getOutputEnable** () const
- void **setOutputEnable** (bool val)
- bool **getOutputEnableAvailable** () const
- bool **getSpecialPurposeOut** () const
- void **setSpecialPurposeOut** (bool val)
- bool **getSpecialPurposeOutAvailable** () const
- bool **getGateOut** () const
- void **setGateOut** (bool val)
- bool **getBuTiSMultiplexer** () const
- void **setBuTiSMultiplexer** (bool val)
- bool **getPPSMultiplexer** () const
- void **setPPSMultiplexer** (bool val)
- bool **StartClock** (double high_phase, double low_phase, uint64_t phase_offset)
- bool **StopClock** ()
- std::string **getLogicLevelOut** () const
- std::string **getTypeOut** () const
- bool **ReadInput** ()
- bool **getInputTermination** () const
- void **setInputTermination** (bool val)
- bool **getInputTerminationAvailable** () const
- bool **getSpecialPurposeIn** () const
- void **setSpecialPurposeIn** (bool val)
- bool **getSpecialPurposeInAvailable** () const
- bool **getGateIn** () const
- void **setGateIn** (bool val)
- std::string **getLogicLevelIn** () const
- std::string **getTypeIn** () const
- uint64_t **getResolution** () const
- bool **ConfigureClock** (double high_phase, double low_phase, uint64_t phase_offset)
- std::string **getLogicLevel** () const
- std::string **getType** () const

Public Attributes

- std::function< void(bool) > **OutputEnable**
- std::function< void(bool) > **SpecialPurposeOut**
- std::function< void(bool) > **GateOut**
- std::function< void(bool) > **BuTiSMultiplexer**
- std::function< void(bool) > **PPSMultiplexer**
- std::function< void(bool) > **InputTermination**
- std::function< void(bool) > **SpecialPurposeIn**
- std::function< void(bool) > **GateIn**

7.74.1 Detailed Description

representaion of a single IO on a TimingReceiver

On the hardware all IOs are controlled IO_CONTROL sdb device which is represented by IoControl class in saftlib. This Io class is used by the IoControl class to represent an individual IO. It encapsulates the register accesses to setup the hardware parameters for the represented IO.

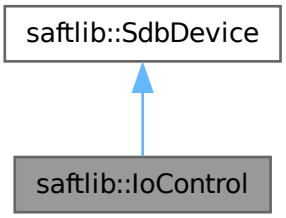
This class is also used by the Input and Output classes, which are both part of the user facing API.

The documentation for this class was generated from the following files:

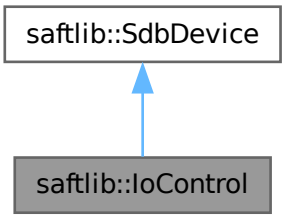
- src/Io.hpp
- src/Io.cpp

7.75 saftlib::IoControl Class Reference

Inheritance diagram for saftlib::IoControl:



Collaboration diagram for saftlib::IoControl:



Public Member Functions

- **IoControl** (etherbone::Device &device)
- std::vector< Io > & **get_ios** ()

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

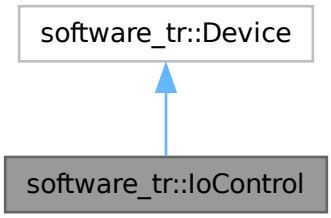
- eb_address_t **adr_first**
- etherbone::Device & **device**

The documentation for this class was generated from the following files:

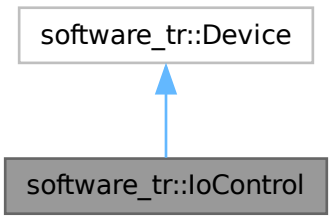
- src/IoControl.hpp
- src/IoControl.cpp

7.76 software_tr::IoControl Class Reference

Inheritance diagram for software_tr::IoControl:



Collaboration diagram for software_tr::IoControl:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0x10c05791 }

Public Member Functions

- **IoControl** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool read_access (uint32_t adr, int sel, uint32_t *dat_out)

Public Member Functions inherited from software_tr::Device

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.76.1 Member Function Documentation

7.76.1.1 contains()

```
bool software_tr::IoControl::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.76.1.2 read_access()

```
bool software_tr::IoControl::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

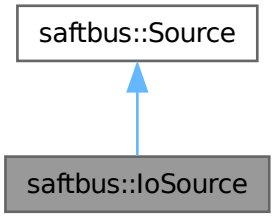
- src/saft-software-tr.cpp

7.77 saftbus::IoSource Class Reference

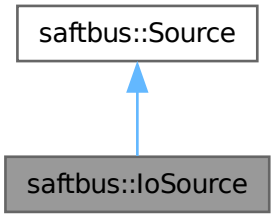
An event source that is ready whenever a certain condition on a given file descriptor is met (usually POLLIN or POLLOUT)

`#include <loop.hpp>`

Inheritance diagram for saftbus::IoSource:



Collaboration diagram for saftbus::IoSource:



Public Member Functions

- **IoSource** (std::function< bool(int, int)> slot, int fd, int condition)
- bool prepare (std::chrono::milliseconds &timeout_ms) override
- bool check () override
- bool dispatch () override
- std::string type () override

Public Member Functions inherited from saftbus::Source

- virtual bool **prepare** (std::chrono::milliseconds &timeout_ms)=0
- virtual bool **check** ()=0
- virtual bool **dispatch** ()=0
- virtual std::string **type** ()=0
- long **get_id** ()

Friends

- class **Loop**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Source

- void **add_poll** (pollfd *pfd)
- void **remove_poll** (pollfd *pfd)
- void **clear_poll** ()

7.77.1 Detailed Description

An event source that is ready whenever a certain condition on a given file descriptor is met (usually POLLIN or POLLOUT)

The source is removed from the loop whenever the callback function returns false or if the file descriptor reports hung-up.

Parameters

slot	the function that is called whenever the file descriptor fulfills the given condition. If it returns true, Source stays active. Otherwise the source is removed from the event loop.
fd	the file descriptor that is observed by the source
condition	the condition that is used to activate the source (usually POLLIN or POLLOUT)

7.77.2 Member Function Documentation

7.77.2.1 check()

```
bool saftbus::IoSource::check ( )  [override], [virtual]
```

Implements saftbus::Source.

7.77.2.2 dispatch()

```
bool saftbus::IoSource::dispatch ( )  [override], [virtual]
```

Implements saftbus::Source.

7.77.2.3 prepare()

```
bool saftbus::IoSource::prepare (
            std::chrono::milliseconds & timeout_ms )  [override], [virtual]
```

Implements saftbus::Source.

7.77.2.4 type()

```
std::string saftbus::IoSource::type ( )  [override], [virtual]
```

Implements saftbus::Source.

The documentation for this class was generated from the following files:

- saftbus/loop.hpp
- saftbus/loop.cpp

7.78 saftlib::IRQ Class Reference

Represents an IRQ that is managed by saftlib.

```
#include <SAFTd.hpp>
```

Public Member Functions

- eb_address_t **address** ()
  address to trigger the IRQ

Friends

- class **SAFTd**

7.78.1 Detailed Description

Represents an IRQ that is managed by saftlib.

A std::unique_ptr<IRQ> is returend by SAFTd::request_irq when passing an MsiDevice to it. The IRQ::address() function returns the wishbone address that the MsiDevice (on the hardware)needs to write to in order to call the attached callback function on the host system. The IRQ destructor automatially releases the irq slot from SAFTd.

The documentation for this class was generated from the following files:

- src/SAFTd.hpp
- src/SAFTd.cpp

7.79 saftbus::LibraryLoader Class Reference

The class is constructed with the name of a shared library file (.so). It opens the file, gets the address of the function symbol ¨create_services¨. The public member function create_services forwards all arguments to the loaded function from the shared library.

```
#include <plugins.hpp>
```

Classes

- struct Impl

Public Member Functions

- **LibraryLoader** (const std::string &so_filename)
- void create_services (Container *container, const std::vector< std::string > &args=std::vector< std::string >())

  call the create_service function of the shared library associated with this object (the so_filename passed to the constructor).

7.79.1 Detailed Description

The class is constructed with the name of a shared library file (.so). It opens the file, gets the address of the function symbol ¨create_services¨. The public member function create_services forwards all arguments to the loaded function from the shared library.

The create_services function in the shared library can use the container pointer to add Service objects into the container.

7.79.2 Member Function Documentation

7.79.2.1 create_services()

```
void saftbus::LibraryLoader::create_services (
            Container * container,
            const std::vector< std::string > & args = std::vector<std::string>() )
```

call the create_service function of the shared library associated with this object (the so_filename passed to the constructor).

Parameters

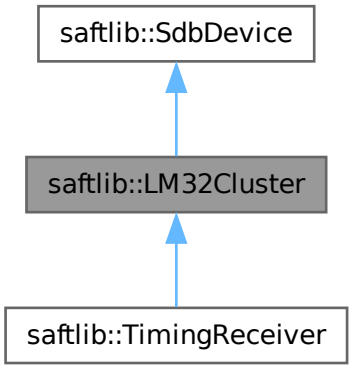
container	normally, the create_service function in the shared library will use this pointer to add Service objects into the container.
args	a vector of string arguments. The meaning of the arguments depends on the shared library. the service objects are owned by the plugin, we only get a naked pointer to it the std::string is the object path under which the respective Service object should be mounted

The documentation for this class was generated from the following files:

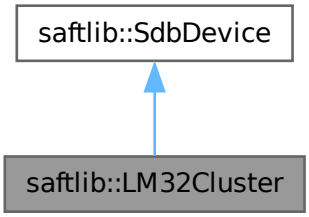
- saftbus/plugins.hpp
- saftbus/plugins.cpp

7.80 saftlib::LM32Cluster Class Reference

Inheritance diagram for saftlib::LM32Cluster:



Collaboration diagram for saftlib::LM32Cluster:



Public Member Functions

- **LM32Cluster** (etherbone::Device &device, TimingReceiver *tr)
- unsigned getCpuCount ()

  number of CPUs
- void SafeHaltCpu (unsigned cpu_idx)

  stop execution of cpu[cpu_idx]
- void **WriteFirmware** (unsigned cpu_idx, const std::string &filename)

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_found=true)

Public Attributes

- std::vector< eb_address_t > **dpram_lm32_adr_first**
- std::vector< eb_address_t > **dpram_lm32_adr_last**

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.80.1 Member Function Documentation

7.80.1.1 getCpuCount()

```
unsigned saftlib::LM32Cluster::getCpuCount ( )
```

number of CPUs

Returns

number of instanciated User LM32 Cores

7.80.1.2 SafeHaltCpu()

```
void saftlib::LM32Cluster::SafeHaltCpu (
            unsigned cpu_idx )
```

stop execution of cpu[cpu_idx]

Parameters

cpu_idx	identifies the cpu that should be halted

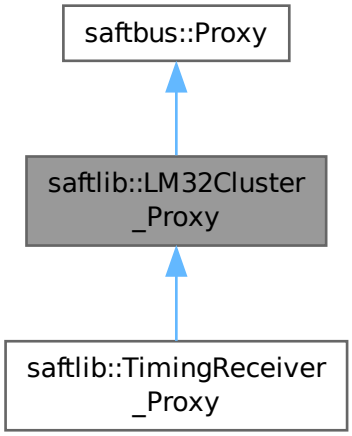
To avoid halting the cpu inside a whishbone cycle, the ram is filled with jump instructions that jump to the same location.

The documentation for this class was generated from the following files:

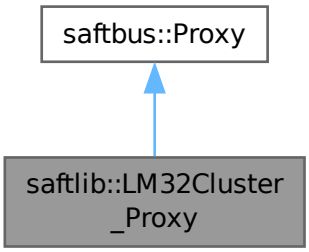
- src/LM32Cluster.hpp
- src/LM32Cluster.cpp

7.81 saftlib::LM32Cluster_Proxy Class Reference

Inheritance diagram for saftlib::LM32Cluster_Proxy:



Collaboration diagram for saftlib::LM32Cluster_Proxy:



Public Member Functions

- **LM32Cluster_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- unsigned getCpuCount ()

  number of CPUs
- void SafeHaltCpu (unsigned cpu_idx)

  stop execution of cpu[cpu_idx]

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< LM32Cluster_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.81.1 Member Function Documentation

7.81.1.1 getCpuCount()

```
unsigned saftlib::LM32Cluster_Proxy::getCpuCount ( )
```

number of CPUs

Returns

number of instanciated User LM32 Cores

7.81.1.2 SafeHaltCpu()

```
void saftlib::LM32Cluster_Proxy::SafeHaltCpu (
            unsigned cpu_idx )
```

stop execution of cpu[cpu_idx]

Parameters

cpu_idx	identifies the cpu that should be halted

To avoid halting the cpu inside a whishbone cycle, the ram is filled with jump instructions that jump to the same location.

7.81.1.3 signal_dispatch()

```
bool saftlib::LM32Cluster_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

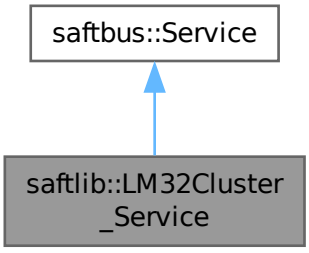
Reimplemented in saftlib::TimingReceiver_Proxy.

The documentation for this class was generated from the following files:

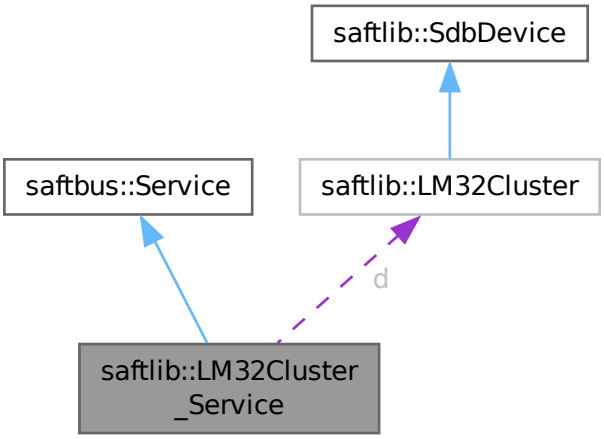
- src/LM32Cluster_Proxy.hpp
- src/LM32Cluster_Proxy.cpp

7.82 saftlib::LM32Cluster_Service Class Reference

Inheritance diagram for saftlib::LM32Cluster_Service:



Collaboration diagram for saftlib::LM32Cluster_Service:



Public Types

- typedef LM32Cluster **DriverType**

Public Member Functions

- **LM32Cluster_Service** (LM32Cluster ∗instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- LM32Cluster * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.82.1 Member Function Documentation

7.82.1.1 call()

```
void saftlib::LM32Cluster_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

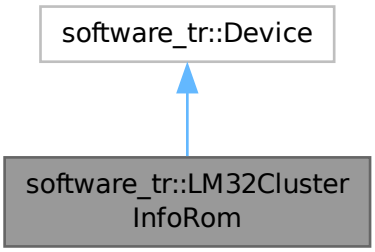
Implements saftbus::Service.

The documentation for this class was generated from the following files:

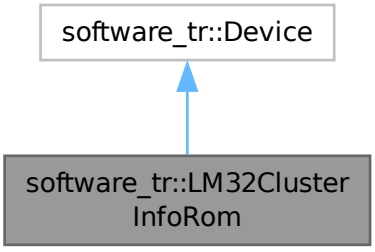
- src/LM32Cluster_Service.hpp
- src/LM32Cluster_Service.cpp
- src/LM32Cluster_Service_old.cpp

7.83 software_tr::LM32ClusterInfoRom Class Reference

Inheritance diagram for software_tr::LM32ClusterInfoRom:



Collaboration diagram for software_tr::LM32ClusterInfoRom:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0x10040086 }

Public Member Functions

- **LM32ClusterInfoRom** (uint32_t adr_first, uint32_t instance)
- bool contains (uint32_t adr)
- bool write_access (uint32_t adr, int sel, uint32_t dat)
- bool read_access (uint32_t adr, int sel, uint32_t *dat_out)

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.83.1 Member Function Documentation

7.83.1.1 contains()

```
bool software_tr::LM32ClusterInfoRom::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.83.1.2 read_access()

```
bool software_tr::LM32ClusterInfoRom::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

7.83.1.3 write_access()

```
bool software_tr::LM32ClusterInfoRom::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

- src/saft-software-tr.cpp

7.84 saftbus::Loop Class Reference

an event loop, driven by Sources

```
#include <loop.hpp>
```

Classes

- struct Impl

Public Member Functions

- bool **iteration** (bool may_block)
- void **run** ()
- bool **quit** ()
- bool **quit_in** (std::chrono::milliseconds wait_ms)
- SourceHandle **connect** (std::unique_ptr< Source > source)
- template<typename T , typename... Args>
  SourceHandle **connect** (Args &&... args)
- void remove (SourceHandle s)

  public version of remove which works with a SourceHandle
- void **clear** ()

Static Public Member Functions

- static Loop & **get_default** ()

7.84.1 Detailed Description

an event loop, driven by Sources

One loop iteration goes like this:

- find earliest timeout from all sources by calling the Source::prepare function
- collect all source file descriptors that need to be polled
- in case there are any file descriptors, do the poll system call
- in case there are no file descriptors, wait until the earliest timeout
- call Source::dispatch for all sources where Source::check returns true.

7.84.2 Member Function Documentation

7.84.2.1 remove()

```
void saftbus::Loop::remove (
            SourceHandle s )
```

public version of remove which works with a SourceHandle

Parameters

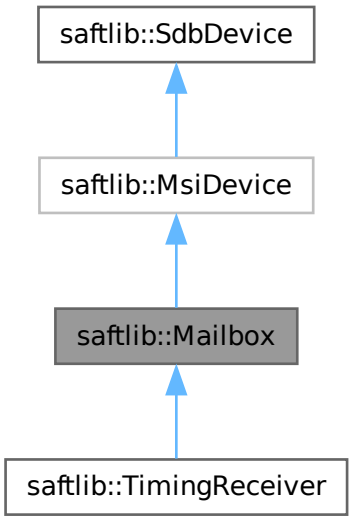
s	the source handle returned from the connect method

The documentation for this class was generated from the following files:

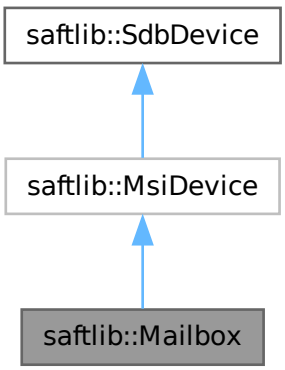
- saftbus/loop.hpp
- saftbus/loop.cpp

7.85 saftlib::Mailbox Class Reference

Inheritance diagram for saftlib::Mailbox:



Collaboration diagram for saftlib::Mailbox:



Classes

- class Slot

Public Member Functions

- void UseSlot (int slot_index, uint32_t value)

  write a value to the preconfigured address.
- **Mailbox** (etherbone::Device &device)
- std::unique_ptr< Slot > ConfigureSlot (uint32_t target_address)

  find a free slot in the mailbox and configure it with target_address

Public Member Functions inherited from saftlib::MsiDevice

- **MsiDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID)

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_found=true)

Friends

- class **Slot**

Additional Inherited Members

Protected Attributes inherited from saftlib::MsiDevice

- etherbone::sdb_msi_device **msi_device**

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.85.1 Member Function Documentation

7.85.1.1 ConfigureSlot()

```
std::unique_ptr< Mailbox::Slot > saftlib::Mailbox::ConfigureSlot (
            uint32_t target_address )
```

find a free slot in the mailbox and configure it with target_address

Parameters

target_address	specifies to which address the value in UseSlot will be written

Returns

the slot number which was used, -1 if no free slot was found

7.85.1.2 UseSlot()

```
void saftlib::Mailbox::UseSlot (
             int slot_index,
             uint32_t value )
```

write a value to the preconfigured address.

Parameters

slot_index	the slot to be used (return value of ConfigureSlot)
value	is the value to be written

The documentation for this class was generated from the following files:

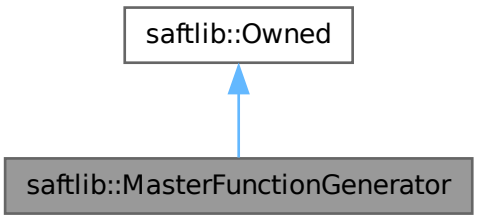
- src/Mailbox.hpp
- src/Mailbox.cpp

7.86 saftlib::MasterFunctionGenerator Class Reference

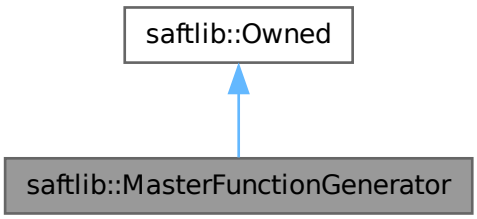
Interface to multiple Function Generators.

```
#include <MasterFunctionGenerator.hpp>
```

Inheritance diagram for saftlib::MasterFunctionGenerator:



Collaboration diagram for saftlib::MasterFunctionGenerator:



Public Member Functions

- MasterFunctionGenerator (saftbus::Container *container, const std::string &object_path, std::vector< std::shared_ptr< FunctionGeneratorImpl > > functionGenerators)
- void Arm ()

  Enable all function generators that have data and arm them.
- void Abort (bool wait_for_abort_ack)

  Abort waveform generation in all function generators.
- void InitializeSharedMemory (const std::string &shared_memory_name)

  Initialze a boost managed_shared_memory region.
- void **AppendParameterTuplesForBeamProcess** (int beam_process, bool arm, bool wait_for_arm_ack)
- bool AppendParameterSets (const std::vector< std::vector< int16_t > > &coeff_a, const std::vector< std::vector< int16_t > > &coeff_b, const std::vector< std::vector< int32_t > > &coeff_c, const std::vector< std::vector< unsigned char > > &step, const std::vector< std::vector< unsigned char > > &freq, const std::vector< std::vector< unsigned char > > &shift_a, const std::vector< std::vector< unsigned char > > &shift_b, bool arm, bool wait_for_arm_ack)

  For each function generator, append parameter tuples describing.
- std::vector< uint32_t > ReadExecutedParameterCounts ()

  Number of parameter tuples executed by each function generator.
- std::vector< uint64_t > ReadFillLevels ()

  Remaining waveform data in nanoseconds for each FG.
- void Flush ()

  Empty the parameter tuple set of all function generators.
- void setStartTag (uint32_t val)

  The SCUbus tag which causes function generation to begin.
- uint32_t **getStartTag** () const
- void setGenerateIndividualSignals (bool newvalue)

  If true, signals from the individual.
- bool **getGenerateIndividualSignals** () const
- std::vector< std::string > ReadAllNames ()

  Read the name for each available FG.
- std::vector< std::string > ReadNames ()

  Read the name for each active FG.
- std::vector< int > ReadArmed ()

  Read the armed state of each active FG.
- std::vector< int > ReadEnabled ()

Read the enabled state of each active FG.

- std::vector< int > ReadRunning ()

  Read the running state of each active FG.

- void ResetActiveFunctionGenerators ()

  Resets the list of active function generators handled by this master to all available FGs.

- void SetActiveFunctionGenerators (const std::vector< std::string > &names)

  Set the list of active function generators handled by this master to the names given.

- std::string **getObjectPath** ()

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.

- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)

- void Disown ()

  Release ownership of the object.

- void Own ()

  Claim ownership of the object.

- std::string getOwner () const

  The client which owns this object.

- bool getDestructible () const

  Can the object be destroyed.

- void Destroy ()

  Destroy this object.

Public Attributes

- sigc::signal< void, std::string, saftlib::Time, bool, bool, bool > SigStopped

  Stopped signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, bool > **Armed**

  Armed signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, bool > **Enabled**

  Enabled signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, bool > **Running**

  Running signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, saftlib::Time > **SigStarted**

  Started signal forwarded from a single Function Generator.

- sigc::signal< void, std::string > **Refill**

  Refill signal forwarded from a single Function Generator.

- sigc::signal< void, saftlib::Time > SigAllStopped

  All Function generators have stopped.

- sigc::signal< void > AllArmed

  All Function generators that have a FillLevel>0 are armed.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions

- void **arm_all** ()
- void **reset_all** ()
- void **ownerQuit** ()
- void **on_fg_running** (std::shared_ptr< FunctionGeneratorImpl > &fg, bool)
- void **on_fg_armed** (std::shared_ptr< FunctionGeneratorImpl > &fg, bool)
- void **on_fg_enabled** (std::shared_ptr< FunctionGeneratorImpl > &fg, bool)
- void **on_fg_started** (std::shared_ptr< FunctionGeneratorImpl > &fg, uint64_t)
- void **on_fg_stopped** (std::shared_ptr< FunctionGeneratorImpl > &fg, uint64_t time, bool abort, bool hardwareUnderflow, bool microcontrollerUnderflow)
- void **on_fg_refill** (std::shared_ptr< FunctionGeneratorImpl > &fg)
- bool **all_armed** ()
- bool **all_stopped** ()
- bool **WaitTimeout** ()
- void **waitForCondition** (std::function< bool()> condition, int timeout_ms)

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes

- std::string **objectPath**
- std::vector< std::shared_ptr< FunctionGeneratorImpl > > **allFunctionGenerators**
- std::vector< std::shared_ptr< FunctionGeneratorImpl > > **activeFunctionGenerators**
- uint32_t **startTag**
- bool **generateIndividualSignals**
- sigc::connection **waitTimeout**
- std::map< int, std::vector< ParameterTuple > > **parametersForBeamProcess**
- std::unique_ptr< boost::interprocess::managed_shared_memory > **shm_params**
- boost::interprocess::interprocess_mutex * **shm_mutex**
- std::map< std::string, ParameterVector * > **paramVectors**

7.86.1 Detailed Description

Interface to multiple Function Generators.

Operation of function generators is aggregated to reduce the number of d-bus operations required.

7.86.2 Constructor & Destructor Documentation

7.86.2.1 MasterFunctionGenerator()

```
saftlib::MasterFunctionGenerator::MasterFunctionGenerator (
            saftbus::Container * container,
            const std::string & object_path,
            std::vector< std::shared_ptr< FunctionGeneratorImpl > > functionGenerators )
```

Parameters

object_path	const std::string &fg_name,

7.86.3 Member Function Documentation

7.86.3.1 Abort()

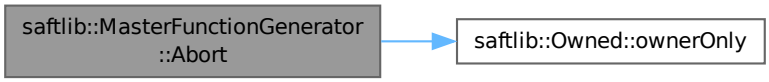
```
void saftlib::MasterFunctionGenerator::Abort (
            bool wait_for_abort_ack )
```

Abort waveform generation in all function generators.

Parameters

wait_for_abort_ack	If true, wait for all FGs to transition to the disabled state before returning.

This directs the hardware to stop waveform generation. If the function generator was Armed, it is disarmed and disabled, without outputting any waveform data. If the function generator is running, output is Stopped at the current value and disabled. Aborting a function generator takes time, so even after a call to Abort, the function generator might still be Started. However, it will reach the disabled state as quickly as it can, transitioning through Stopped as usual. If the Owner of a FunctionGenerator quits without running Disown, the Abort is run automatically. Here is the call graph for this function:



7.86.3.2 AppendParameterSets()

```
bool saftlib::MasterFunctionGenerator::AppendParameterSets (
            const std::vector< std::vector< int16_t > > & coeff_a,
            const std::vector< std::vector< int16_t > > & coeff_b,
            const std::vector< std::vector< int32_t > > & coeff_c,
            const std::vector< std::vector< unsigned char > > & step,
            const std::vector< std::vector< unsigned char > > & freq,
            const std::vector< std::vector< unsigned char > > & shift_a,
            const std::vector< std::vector< unsigned char > > & shift_b,
            bool arm,
            bool wait_for_arm_ack )
```

For each function generator, append parameter tuples describing.

the waveform to generate. Each parameter is sent as a vector of vectors: per FG then per tuple element

Parameters are sent as a vector of vectors. The outside vectors contain a coefficient vector for each FG and must be of the same size and less or equal the number of active FGs. The coefficient vectors for each FG's parameter set must be the same size but different FGs may have different parameter set sizes. If there is no data for an individual function generator an empty vector should be sent.

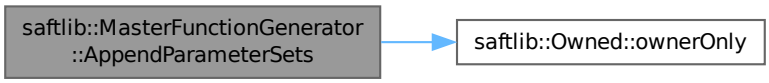
@coeff_a: Quadratic coefficient (a), 16-bit signed @coeff_b: Linear coefficient (b), 16-bit signed @coeff_c↩ : Constant coefficient (c), 32-bit signed @step: Number of interpolated samples (0=250, 1=500, 2=1000, 3=2000, 4=4000, 5=8000, 6=16000, or 7=32000) @freq: Output sample frequency (0=16kHz, 1=32kHz, 2=64kHz, 3=125k↩ Hz, 4=250kHz, 5=500kHz, 6=1MHz, or 7=2MHz) @shift_a: Exponent of coeff_a, 6-bit unsigned; a = coeff_↩ a∗2^shift_a @shift_b: Exponent of coeff_b, 6-bit unsigned; b = coeff_b∗2^shift_b

@arm: If true, arm each function generator that received data and wait for acknowledgement @wait_for_arm_ack: If true, and arm is true, wait for arm acknowledgements from all fgs before returning @low_fill: Fill level remains low for at least one FG - use ReadFillLevels

This function appends the parameter vectors (which must be equal in length) to the FIFO of remaining waveform to generate. Each parameter set (coefficients) describes a number of output samples in the generated wave form. Parameter sets are executed in order until no more remain.

If the fill level is not high enough, this method returns true. Only once this function has returned false can you await the Refill signal.

At each step, the function generator outputs high_bits(c∗2^32 + b∗t + c∗t∗t), where t ranges from 0 to numSteps-1. high_bits are the high OutputWindowSize bits of the resulting 64-bit signed value. Here is the call graph for this function:



7.86.3.3 Arm()

```
void saftlib::MasterFunctionGenerator::Arm ( )
```

Enable all function generators that have data and arm them.

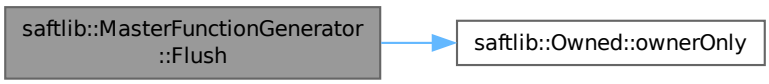
A function generator can only be Armed if FillLevel is non-zero. An Enabled function generator can not be Armed again until it either completes waveform generation or the user calls Abort. Arming a function generator takes time. Wait for Armed to transition to true before sending StartTag. Here is the call graph for this function:

7.86.3.4 Flush()

```
void saftlib::MasterFunctionGenerator::Flush ( )
```

Empty the parameter tuple set of all function generators.

Flush may only be called when not Enabled. Flush does not clear the ExecutedParameterCount. Here is the call graph for this function:



7.86.3.5 InitializeSharedMemory()

```
void saftlib::MasterFunctionGenerator::InitializeSharedMemory (
            const std::string & shared_memory_name )
```

Initialze a boost managed_shared_memory region.

It should contain: mutex ParameterVector

7.86.3.6 ReadAllNames()

```
std::vector< std::string > saftlib::MasterFunctionGenerator::ReadAllNames ( )
```

Read the name for each available FG.

Returns

Name of each available FG.

7.86.3.7 ReadArmed()

```
std::vector< int > saftlib::MasterFunctionGenerator::ReadArmed ( )
```

Read the armed state of each active FG.

Returns

State of each FG.

7.86.3.8 ReadEnabled()

```
std::vector< int > saftlib::MasterFunctionGenerator::ReadEnabled ( )
```

Read the enabled state of each active FG.

Returns

State of each FG.

7.86.3.9 ReadExecutedParameterCounts()

```
std::vector< uint32_t > saftlib::MasterFunctionGenerator::ReadExecutedParameterCounts ( )
```

Number of parameter tuples executed by each function generator.

This counts the total number of parameter tuples executed since the last Started signal. Obviously, if the function generator is running, the returned value will be old.

Returns

Number tuples executed by hardware.

7.86.3.10 ReadFillLevels()

```
std::vector< uint64_t > saftlib::MasterFunctionGenerator::ReadFillLevels ( )
```

Remaining waveform data in nanoseconds for each FG.

The SAFTd has sufficient parameters buffered to supply the function generator with data for the specified time in nanoseconds. Note, due to the slow nature of software, if the function generator is currently running, the read value will already be out-of-date upon return. This property should be used for informational use only.

Returns

Remaining waveform data in nanoseconds for each FG.

7.86.3.11 ReadNames()

```
std::vector< std::string > saftlib::MasterFunctionGenerator::ReadNames ( )
```

Read the name for each active FG.

Returns

Name of each active FG.

7.86.3.12 ReadRunning()

```
std::vector< int > saftlib::MasterFunctionGenerator::ReadRunning ( )
```

Read the running state of each active FG.

Returns

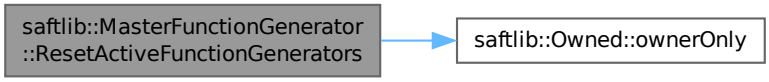
State of each FG.

7.86.3.13 ResetActiveFunctionGenerators()

```
void saftlib::MasterFunctionGenerator::ResetActiveFunctionGenerators ( )
```

Resets the list of active function generators handled by this master to all available FGs.

Here is the call graph for this function:



7.86.3.14 SetActiveFunctionGenerators()

```
void saftlib::MasterFunctionGenerator::SetActiveFunctionGenerators (
            const std::vector< std::string > & names )
```

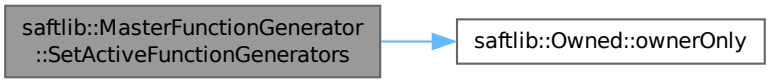
Set the list of active function generators handled by this master to the names given.

Format: fg-[SCUBusSlot]-[DeviceNumber]-[index] e.g. {"fg-3-0-0","fg-3-1-1"}

Parameters

names	List of names identifying each FG to activate.

Here is the call graph for this function:



7.86.3.15 setGenerateIndividualSignals()

```
void saftlib::MasterFunctionGenerator::setGenerateIndividualSignals (
            bool newvalue )
```

If true, signals from the individual.

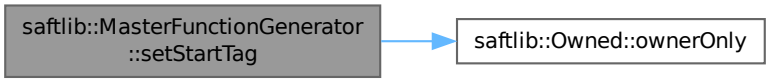
function generators are forwarded. The aggregate signals will still be generated. This defaults to false to reduce the load on the d-bus message bus.

7.86.3.16 setStartTag()

```
void saftlib::MasterFunctionGenerator::setStartTag (
            uint32_t val )
```

The SCUbus tag which causes function generation to begin.

All function generators under control of the Master use the same tag. If the function generator is Armed and this tag is sent to the SCUbus, then the function generator will begin generating the output waveform. StartTag may only be set when the FunctionGenerator is not Enabled. Here is the call graph for this function:



7.86.4 Member Data Documentation

7.86.4.1 AllArmed

```
sigc::signal< void > saftlib::MasterFunctionGenerator::AllArmed
```

All Function generators that have a FillLevel>0 are armed.

This signal is generated when a function generator signals that it is armed, and all function generators controlled by this master that have FillLevel>0 are armed.

7.86.4.2 SigAllStopped

`sigc::signal< void , saftlib::Time > saftlib::MasterFunctionGenerator::SigAllStopped`

All Function generators have stopped.

This signal is generated when a function generator stops and all function generators controlled by this master have then stopped.

Parameters

time	Time when last function generation ended

7.86.4.3 SigStopped

`sigc::signal< void , std::string , saftlib::Time , bool , bool , bool > saftlib::Master↩FunctionGenerator::SigStopped`

Stopped signal forwarded from a single Function Generator.

Parameters

name	Name of the FG that generated the signal.
time	Time when function generation ended in nanoseconds since 1970
abort	stopped due to a call to Abort
hardwareMacroUnderflow	A fatal error, indicating the SCUbus is congested
microControllerUnderflow	A fatal error, indicating the host CPU is overloaded

The function generator stops either successfully (when all data has been sent), or it stops due to an error. When an error occurs, the function generator stops and holds its most recent value. This can occur due to two causes:

hardwareMacroUnderflow, a fatal error indicating the hardware ran out of data. If the SCUbus is too busy, it can happen that the waveform data stored in the function generator HDL is not refilled in time. This error can only be mitigated by ensuring that the function generator does not share the SCUbus with other users.

microControllerUnderflow, a fatal error indicating the microcontroller ran out of data. If the host CPU running this software is too busy, it can happen that the waveform data is not delivered to the microcontroller before the microcontroller runs out of data. This error can be mitigated by reducing the number of busy processes running on the system.

Once the function generator has stopped, ExecutedParameterCount remains valid until the next time the function generator starts. After stopping, regardless of if the generation was successful or not, the parameter FIFO is cleared, Enabled is false, and this signal emitted.

The documentation for this class was generated from the following files:

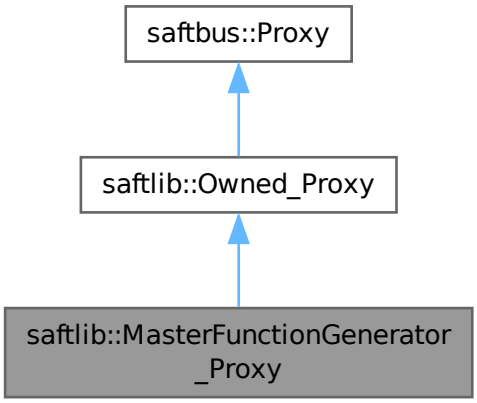
- src/MasterFunctionGenerator.hpp
- src/MasterFunctionGenerator.cpp

7.87 saftlib::MasterFunctionGenerator_Proxy Class Reference

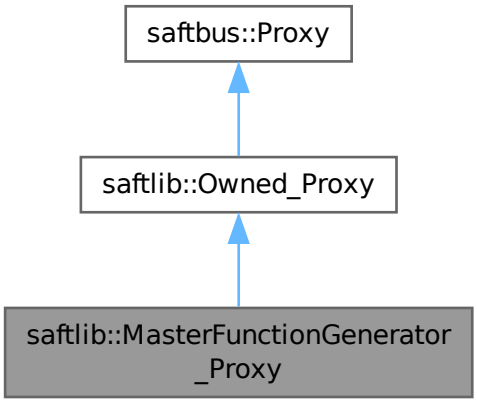
Interface to multiple Function Generators.

`#include <MasterFunctionGenerator_Proxy.hpp>`

Inheritance diagram for saftlib::MasterFunctionGenerator_Proxy:



Collaboration diagram for saftlib::MasterFunctionGenerator_Proxy:



Public Member Functions

- **MasterFunctionGenerator_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_↩ group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_↩ interface_names())

- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Arm ()

  Enable all function generators that have data and arm them.
- void Abort (bool wait_for_abort_ack)

  Abort waveform generation in all function generators.
- void InitializeSharedMemory (const std::string &shared_memory_name)

  Initialze a boost managed_shared_memory region.
- void **AppendParameterTuplesForBeamProcess** (int beam_process, bool arm, bool wait_for_arm_ack)
- bool AppendParameterSets (const std::vector< std::vector< int16_t > > &coeff_a, const std::vector< std::vector< int16_t > > &coeff_b, const std::vector< std::vector< int32_t > > &coeff_c, const std::vector< std::vector< unsigned char > > &step, const std::vector< std::vector< unsigned char > > &freq, const std::vector< std::vector< unsigned char > > &shift_a, const std::vector< std::vector< unsigned char > > &shift_b, bool arm, bool wait_for_arm_ack)

  For each function generator, append parameter tuples describing.
- std::vector< uint32_t > ReadExecutedParameterCounts ()

  Number of parameter tuples executed by each function generator.
- std::vector< uint64_t > ReadFillLevels ()

  Remaining waveform data in nanoseconds for each FG.
- void Flush ()

  Empty the parameter tuple set of all function generators.
- void setStartTag (uint32_t val)

  The SCUbus tag which causes function generation to begin.
- uint32_t **getStartTag** ()
- void setGenerateIndividualSignals (bool newvalue)

  If true, signals from the individual.
- bool **getGenerateIndividualSignals** ()
- std::vector< std::string > ReadAllNames ()

  Read the name for each available FG.
- std::vector< std::string > ReadNames ()

  Read the name for each active FG.
- std::vector< int > ReadArmed ()

  Read the armed state of each active FG.
- std::vector< int > ReadEnabled ()

  Read the enabled state of each active FG.
- std::vector< int > ReadRunning ()

  Read the running state of each active FG.
- void ResetActiveFunctionGenerators ()

  Resets the list of active function generators handled by this master to all available FGs.
- void SetActiveFunctionGenerators (const std::vector< std::string > &names)

  Set the list of active function generators handled by this master to the names given.

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

Release ownership of the object.

- void Own ()

  Claim ownership of the object.

- std::string getOwner ()

  The client which owns this object.

- bool getDestructible ()

  Can the object be destroyed.

- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< MasterFunctionGenerator_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Public Attributes

- sigc::signal< void, std::string, saftlib::Time, bool, bool, bool > SigStopped

  Stopped signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, bool > **Armed**

  Armed signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, bool > **Enabled**

  Enabled signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, bool > **Running**

  Running signal forwarded from a single Function Generator.

- sigc::signal< void, std::string, saftlib::Time > **SigStarted**

  Started signal forwarded from a single Function Generator.

- sigc::signal< void, std::string > **Refill**

  Refill signal forwarded from a single Function Generator.

- sigc::signal< void, saftlib::Time > SigAllStopped

  All Function generators have stopped.

- sigc::signal< void > AllArmed

  All Function generators that have a FillLevel>0 are armed.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.87.1 Detailed Description

Interface to multiple Function Generators.

Operation of function generators is aggregated to reduce the number of d-bus operations required.

7.87.2 Member Function Documentation

7.87.2.1 Abort()

```
void saftlib::MasterFunctionGenerator_Proxy::Abort (
            bool wait_for_abort_ack )
```

Abort waveform generation in all function generators.

Parameters

wait_for_abort_ack	If true, wait for all FGs to transition to the disabled state before returning.

This directs the hardware to stop waveform generation. If the function generator was Armed, it is disarmed and disabled, without outputting any waveform data. If the function generator is running, output is Stopped at the current value and disabled. Aborting a function generator takes time, so even after a call to Abort, the function generator might still be Started. However, it will reach the disabled state as quickly as it can, transitioning through Stopped as usual. If the Owner of a FunctionGenerator quits without running Disown, the Abort is run automatically.

7.87.2.2 AppendParameterSets()

```
bool saftlib::MasterFunctionGenerator_Proxy::AppendParameterSets (
            const std::vector< std::vector< int16_t > > & coeff_a,
            const std::vector< std::vector< int16_t > > & coeff_b,
            const std::vector< std::vector< int32_t > > & coeff_c,
            const std::vector< std::vector< unsigned char > > & step,
            const std::vector< std::vector< unsigned char > > & freq,
            const std::vector< std::vector< unsigned char > > & shift_a,
            const std::vector< std::vector< unsigned char > > & shift_b,
            bool arm,
            bool wait_for_arm_ack )
```

For each function generator, append parameter tuples describing.

the waveform to generate. Each parameter is sent as a vector of vectors: per FG then per tuple element

Parameters are sent as a vector of vectors. The outside vectors contain a coefficient vector for each FG and must be of the same size and less or equal the number of active FGs. The coefficient vectors for each FG's parameter set must be the same size but different FGs may have different parameter set sizes. If there is no data for an individual function generator an empty vector should be sent.

@coeff_a: Quadratic coefficient (a), 16-bit signed @coeff_b: Linear coefficient (b), 16-bit signed @coeff_c↩ : Constant coefficient (c), 32-bit signed @step: Number of interpolated samples (0=250, 1=500, 2=1000, 3=2000, 4=4000, 5=8000, 6=16000, or 7=32000) @freq: Output sample frequency (0=16kHz, 1=32kHz, 2=64kHz, 3=125k↩ Hz, 4=250kHz, 5=500kHz, 6=1MHz, or 7=2MHz) @shift_a: Exponent of coeff_a, 6-bit unsigned; a = coeff_↩ a∗2^shift_a @shift_b: Exponent of coeff_b, 6-bit unsigned; b = coeff_b∗2^shift_b

@arm: If true, arm each function generator that received data and wait for acknowledgement @wait_for_arm_ack: If true, and arm is true, wait for arm acknowledgements from all fgs before returning @low_fill: Fill level remains low for at least one FG - use ReadFillLevels

This function appends the parameter vectors (which must be equal in length) to the FIFO of remaining waveform to generate. Each parameter set (coefficients) describes a number of output samples in the generated wave form. Parameter sets are executed in order until no more remain.

If the fill level is not high enough, this method returns true. Only once this function has returned false can you await the Refill signal.

At each step, the function generator outputs high_bits(c∗2^32 + b∗t + c∗t∗t), where t ranges from 0 to numSteps-1. high_bits are the high OutputWindowSize bits of the resulting 64-bit signed value.

7.87.2.3 Arm()

```
void saftlib::MasterFunctionGenerator_Proxy::Arm ( )
```

Enable all function generators that have data and arm them.

A function generator can only be Armed if FillLevel is non-zero. An Enabled function generator can not be Armed again until it either completes waveform generation or the user calls Abort. Arming a function generator takes time. Wait for Armed to transition to true before sending StartTag.

7.87.2.4 Flush()

```
void saftlib::MasterFunctionGenerator_Proxy::Flush ( )
```

Empty the parameter tuple set of all function generators.

Flush may only be called when not Enabled. Flush does not clear the ExecutedParameterCount.

7.87.2.5 InitializeSharedMemory()

```
void saftlib::MasterFunctionGenerator_Proxy::InitializeSharedMemory (
            const std::string & shared_memory_name )
```

Initialze a boost managed_shared_memory region.

It should contain: mutex ParameterVector

7.87.2.6 ReadAllNames()

```
std::vector< std::string > saftlib::MasterFunctionGenerator_Proxy::ReadAllNames ( )
```

Read the name for each available FG.

Returns

Name of each available FG.

7.87.2.7 ReadArmed()

```
std::vector< int > saftlib::MasterFunctionGenerator_Proxy::ReadArmed ( )
```

Read the armed state of each active FG.

Returns

State of each FG.

7.87.2.8 ReadEnabled()

```
std::vector< int > saftlib::MasterFunctionGenerator_Proxy::ReadEnabled ( )
```

Read the enabled state of each active FG.

Returns

State of each FG.

7.87.2.9 ReadExecutedParameterCounts()

```
std::vector< uint32_t > saftlib::MasterFunctionGenerator_Proxy::ReadExecutedParameterCounts (
)
```

Number of parameter tuples executed by each function generator.

This counts the total number of parameter tuples executed since the last Started signal. Obviously, if the function generator is running, the returned value will be old.

Returns

Number tuples executed by hardware.

7.87.2.10 ReadFillLevels()

```
std::vector< uint64_t > saftlib::MasterFunctionGenerator_Proxy::ReadFillLevels ( )
```

Remaining waveform data in nanoseconds for each FG.

The SAFTd has sufficient parameters buffered to supply the function generator with data for the specified time in nanoseconds. Note, due to the slow nature of software, if the function generator is currently running, the read value will already be out-of-date upon return. This property should be used for informational use only.

Returns

Remaining waveform data in nanoseconds for each FG.

7.87.2.11 ReadNames()

```
std::vector< std::string > saftlib::MasterFunctionGenerator_Proxy::ReadNames ( )
```

Read the name for each active FG.

Returns

Name of each active FG.

7.87.2.12 ReadRunning()

```
std::vector< int > saftlib::MasterFunctionGenerator_Proxy::ReadRunning ( )
```

Read the running state of each active FG.

Returns

State of each FG.

7.87.2.13 ResetActiveFunctionGenerators()

```
void saftlib::MasterFunctionGenerator_Proxy::ResetActiveFunctionGenerators ( )
```

Resets the list of active function generators handled by this master to all available FGs.

7.87.2.14 SetActiveFunctionGenerators()

```
void saftlib::MasterFunctionGenerator_Proxy::SetActiveFunctionGenerators (
            const std::vector< std::string > & names )
```

Set the list of active function generators handled by this master to the names given.

Format: fg-[SCUBusSlot]-[DeviceNumber]-[index] e.g. {¨fg-3-0-0¨,¨fg-3-1-1¨}

Parameters

names	List of names identifying each FG to activate.

7.87.2.15 setGenerateIndividualSignals()

```
void saftlib::MasterFunctionGenerator_Proxy::setGenerateIndividualSignals (
            bool newvalue )
```

If true, signals from the individual.

function generators are forwarded. The aggregate signals will still be generated. This defaults to false to reduce the load on the d-bus message bus.

7.87.2.16 setStartTag()

```
void saftlib::MasterFunctionGenerator_Proxy::setStartTag (
            uint32_t val )
```

The SCUbus tag which causes function generation to begin.

All function generators under control of the Master use the same tag. If the function generator is Armed and this tag is sent to the SCUbus, then the function generator will begin generating the output waveform. StartTag may only be set when the FunctionGenerator is not Enabled.

7.87.2.17 signal_dispatch()

```
bool saftlib::MasterFunctionGenerator_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

7.87.3 Member Data Documentation

7.87.3.1 AllArmed

```
sigc::signal<void> saftlib::MasterFunctionGenerator_Proxy::AllArmed
```

All Function generators that have a FillLevel>0 are armed.

This signal is generated when a function generator signals that it is armed, and all function generators controlled by this master that have FillLevel>0 are armed.

7.87.3.2 SigAllStopped

```
sigc::signal<void, saftlib::Time> saftlib::MasterFunctionGenerator_Proxy::SigAllStopped
```

All Function generators have stopped.

This signal is generated when a function generator stops and all function generators controlled by this master have then stopped.

Parameters

time	Time when last function generation ended

7.87.3.3 SigStopped

```
sigc::signal<void, std::string, saftlib::Time, bool, bool, bool> saftlib::MasterFunction↩
Generator_Proxy::SigStopped
```

Stopped signal forwarded from a single Function Generator.

Parameters

name	Name of the FG that generated the signal.
time	Time when function generation ended in nanoseconds since 1970
abort	stopped due to a call to Abort
hardwareMacroUnderflow	A fatal error, indicating the SCUbus is congested
microControllerUnderflow	A fatal error, indicating the host CPU is overloaded

The function generator stops either successfully (when all data has been sent), or it stops due to an error. When an error occurs, the function generator stops and holds its most recent value. This can occur due to two causes:

hardwareMacroUnderflow, a fatal error indicating the hardware ran out of data. If the SCUbus is too busy, it can happen that the waveform data stored in the function generator HDL is not refilled in time. This error can only be mitigated by ensuring that the function generator does not share the SCUbus with other users.

microControllerUnderflow, a fatal error indicating the microcontroller ran out of data. If the host CPU running this software is too busy, it can happen that the waveform data is not delivered to the microcontroller before the microcontroller runs out of data. This error can be mitigated by reducing the number of busy processes running on the system.

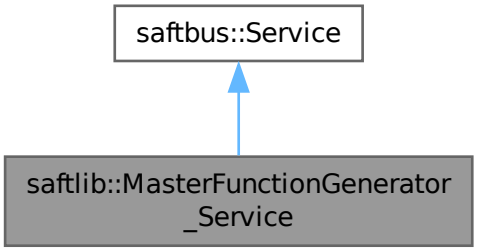
Once the function generator has stopped, ExecutedParameterCount remains valid until the next time the function generator starts. After stopping, regardless of if the generation was successful or not, the parameter FIFO is cleared, Enabled is false, and this signal emitted.

The documentation for this class was generated from the following files:

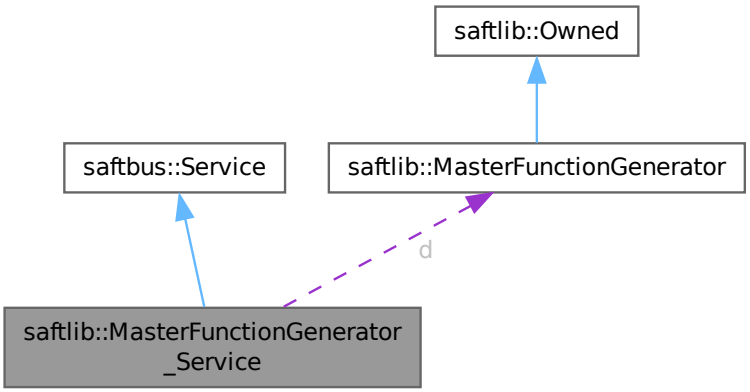
- src/MasterFunctionGenerator_Proxy.hpp
- src/MasterFunctionGenerator_Proxy.cpp

7.88 saftlib::MasterFunctionGenerator_Service Class Reference

Inheritance diagram for saftlib::MasterFunctionGenerator_Service:



Collaboration diagram for saftlib::MasterFunctionGenerator_Service:



Public Types

- typedef MasterFunctionGenerator **DriverType**

Public Member Functions

- **MasterFunctionGenerator_Service** (MasterFunctionGenerator *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

- void **SigStopped_dispatch_function** (std::string arg_1, saftlib::Time arg_2, bool arg_3, bool arg_4, bool arg_5)
- void **Armed_dispatch_function** (std::string arg_1, bool arg_2)
- void **Enabled_dispatch_function** (std::string arg_1, bool arg_2)
- void **Running_dispatch_function** (std::string arg_1, bool arg_2)
- void **SigStarted_dispatch_function** (std::string arg_1, saftlib::Time arg_2)
- void **Refill_dispatch_function** (std::string arg_1)
- void **SigAllStopped_dispatch_function** (saftlib::Time arg_1)
- void **AllArmed_dispatch_function** ()
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- MasterFunctionGenerator * **d**
- sigc::connection **SigStopped_connection**
- sigc::connection **Armed_connection**
- sigc::connection **Enabled_connection**
- sigc::connection **Running_connection**
- sigc::connection **SigStarted_connection**
- sigc::connection **Refill_connection**
- sigc::connection **SigAllStopped_connection**
- sigc::connection **AllArmed_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.88.1 Member Function Documentation

7.88.1.1 call()

```
void saftlib::MasterFunctionGenerator_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

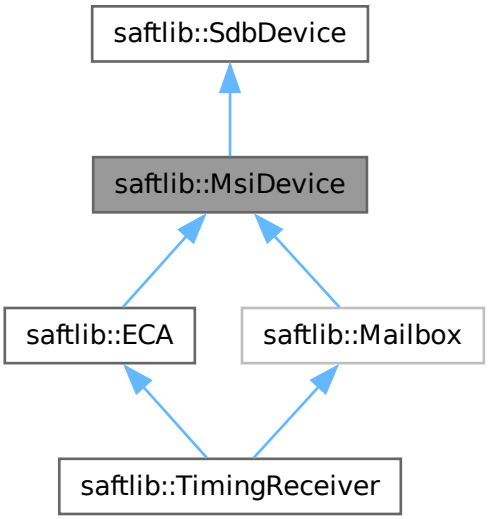
Implements saftbus::Service.

The documentation for this class was generated from the following files:

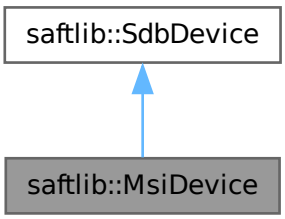
- src/MasterFunctionGenerator_Service.hpp
- src/MasterFunctionGenerator_Service.cpp

7.89 saftlib::MsiDevice Class Reference

Inheritance diagram for saftlib::MsiDevice:



Collaboration diagram for saftlib::MsiDevice:



Public Member Functions

- **MsiDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID)

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Protected Attributes

- etherbone::sdb_msi_device **msi_device**

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

Friends

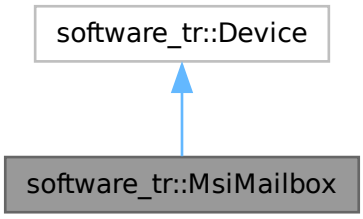
- class **SAFTd**

The documentation for this class was generated from the following files:

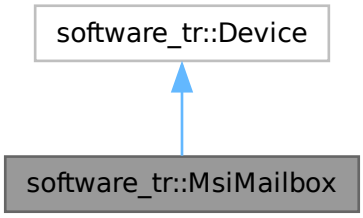
- src/MsiDevice.hpp
- src/MsiDevice.cpp

7.90 software_tr::MsiMailbox Class Reference

Inheritance diagram for software_tr::MsiMailbox:



Collaboration diagram for software_tr::MsiMailbox:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0xfab0bdd8 }

Public Member Functions

- **MsiMailbox** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool write_access (uint32_t adr, int sel, uint32_t dat)
- bool read_access (uint32_t adr, int sel, uint32_t ∗dat_out)

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t ∗dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.90.1 Member Function Documentation

7.90.1.1 contains()

```
bool software_tr::MsiMailbox::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.90.1.2 read_access()

```
bool software_tr::MsiMailbox::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

7.90.1.3 write_access()

```
bool software_tr::MsiMailbox::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

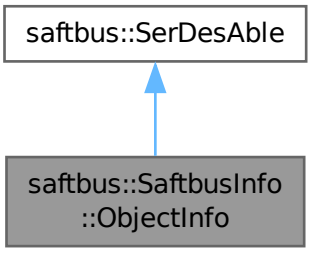
- src/saft-software-tr.cpp

7.91 saftbus::SaftbusInfo::ObjectInfo Struct Reference

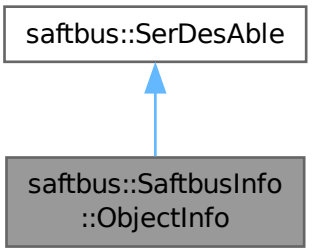
contains all information about a service object

`#include <client.hpp>`

Inheritance diagram for saftbus::SaftbusInfo::ObjectInfo:



Collaboration diagram for saftbus::SaftbusInfo::ObjectInfo:



Public Member Functions

- void serialize (Serializer &ser) const

  custom serializer
- void deserialize (const Deserializer &des)

  custom deserializer

- virtual void serialize (Serializer &ser) const =0
- virtual void deserialize (const Deserializer &des)=0

Public Attributes

- unsigned **object_id**
- std::string **object_path**
- std::vector< std::string > **interface_names**
- std::map< int, int > **signal_fds_use_count**
- int **owner**
- bool **has_destruction_callback**
- bool **destroy_if_owner_quits**

7.91.1 Detailed Description

contains all information about a service object

7.91.2 Member Function Documentation

7.91.2.1 deserialize()

```
void saftbus::SaftbusInfo::ObjectInfo::deserialize (
            const Deserializer & des )  [inline], [virtual]
```

custom deserializer

Implements saftbus::SerDesAble.

7.91.2.2 serialize()

```
void saftbus::SaftbusInfo::ObjectInfo::serialize (
            Serializer & ser ) const  [inline], [virtual]
```

custom serializer

Implements saftbus::SerDesAble.

The documentation for this struct was generated from the following file:

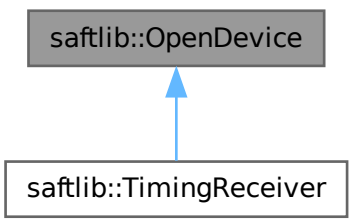
- saftbus/client.hpp

7.92 saftlib::OpenDevice Class Reference

Holds etherbone::Device that is opened on construction and closed on destruction.

`#include <OpenDevice.hpp>`

Inheritance diagram for saftlib::OpenDevice:



Public Member Functions

- OpenDevice (const etherbone::Socket &socket, const std::string &etherbone_path, int polling_interval_ms=1, SAFTd *saftd=nullptr)

  open given etherbone_path on given socket.
- etherbone::Device & **get_device** ()
- std::string getEtherbonePath () const

  The path through which the device is reached.
- std::string getEbForwardPath () const

  If the device is not capable of multiplexing multiple users a /dev/pts/<num> device is created that can be used with eb-tools.

Protected Attributes

- std::string **etherbone_path**
- struct stat **dev_stat**
- etherbone::Device **device**

7.92.1 Detailed Description

Holds etherbone::Device that is opened on construction and closed on destruction.

It remembers its etherbone path and restores the device settings before destruction. It also checks on startup if MSI needs to be polled from the etherbone-slave config regisers or if the hardware has the capability to deliver MSIs without polling The checking procedure for MSI type is:

- Register a MSI callback function for a specific address
- Use the Mailbox device to send an MSI value with that specific address
- start the periodic polling function
- if the polling function is called and finds the specific MSI value, it continues to poll
- if the MSI callback function is called despite of the polling function not seeing the MSI value, the polling function will be removed from the event loop

7.92.2 Constructor & Destructor Documentation

7.92.2.1 OpenDevice()

```
saftlib::OpenDevice::OpenDevice (
            const etherbone::Socket & socket,
            const std::string & etherbone_path,
            int polling_interval_ms = 1,
            SAFTd * saftd = nullptr )
```

open given etherbone_path on given socket.

Parameters

socket	the etherbone Socket
etherbone_path	path of the etherbone device
polling_interval_ms	in case of hardware without native MSIs
saftd	must be a valid pointer if MSIs are used

Here is the call graph for this function:

7.92.3 Member Function Documentation

7.92.3.1 getEbForwardPath()

```
std::string saftlib::OpenDevice::getEbForwardPath ( ) const
```

If the device is not capable of multiplexing multiple users a /dev/pts/<num> device is created that can be used with eb-tools.

Returns

The path which can be used by eb-tools to access the device. If the etherbone device has multiplexing capability no forwarding device is created and this function returns the original etherbone path of the device

7.92.3.2 getEtherbonePath()

```
std::string saftlib::OpenDevice::getEtherbonePath ( ) const
```

The path through which the device is reached.

Returns

The path through which the device is reached.

The documentation for this class was generated from the following files:

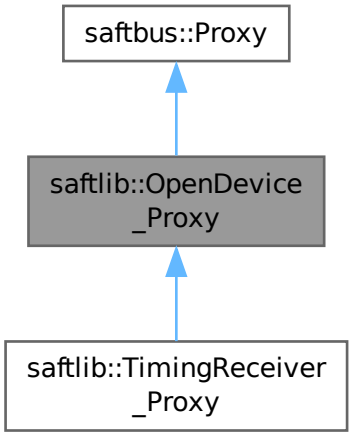
- src/OpenDevice.hpp
- src/OpenDevice.cpp

7.93 saftlib::OpenDevice_Proxy Class Reference

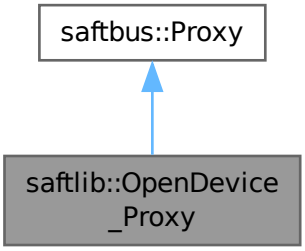
Holds etherbone::Device that is opened on construction and closed on destruction.

`#include <OpenDevice_Proxy.hpp>`

Inheritance diagram for saftlib::OpenDevice_Proxy:



Collaboration diagram for saftlib::OpenDevice_Proxy:



Public Member Functions

- **OpenDevice_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

- std::string getEtherbonePath ()

  The path through which the device is reached.
- std::string getEbForwardPath ()

  If the device is not capable of multiplexing multiple users a /dev/pts/<num> device is created that can be used with eb-tools.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< OpenDevice_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.93.1 Detailed Description

Holds etherbone::Device that is opened on construction and closed on destruction.

It remembers its etherbone path and restores the device settings before destruction. It also checks on startup if MSI needs to be polled from the etherbone-slave config regisers or if the hardware has the capability to deliver MSIs without polling The checking procedure for MSI type is:

- Register a MSI callback function for a specific address
- Use the Mailbox device to send an MSI value with that specific address
- start the periodic polling function
- if the polling function is called and finds the specific MSI value, it continues to poll
- if the MSI callback function is called despite of the polling function not seeing the MSI value, the polling function will be removed from the event loop

7.93.2 Member Function Documentation

7.93.2.1 getEbForwardPath()

```
std::string saftlib::OpenDevice_Proxy::getEbForwardPath ( )
```

If the device is not capable of multiplexing multiple users a /dev/pts/<num> device is created that can be used with eb-tools.

Returns

The path which can be used by eb-tools to access the device. If the etherbone device has multiplexing capability no forwarding device is created and this function returns the original etherbone path of the device

7.93.2.2 getEtherbonePath()

```
std::string saftlib::OpenDevice_Proxy::getEtherbonePath ( )
```

The path through which the device is reached.

Returns

The path through which the device is reached.

7.93.2.3 signal_dispatch()

```
bool saftlib::OpenDevice_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

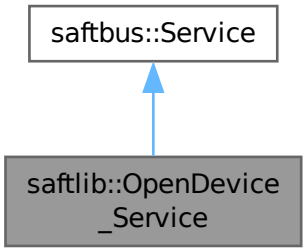
Reimplemented in saftlib::TimingReceiver_Proxy.

The documentation for this class was generated from the following files:

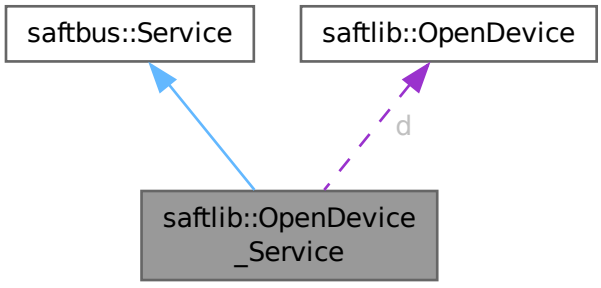
- src/OpenDevice_Proxy.hpp
- src/OpenDevice_Proxy.cpp

7.94 saftlib::OpenDevice_Service Class Reference

Inheritance diagram for saftlib::OpenDevice_Service:



Collaboration diagram for saftlib::OpenDevice_Service:



Public Types

- typedef OpenDevice **DriverType**

Public Member Functions

- **OpenDevice_Service** (OpenDevice *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- OpenDevice * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.94.1 Member Function Documentation

7.94.1.1 call()

```
void saftlib::OpenDevice_Service::call (
             unsigned interface_no,
             unsigned function_no,
             int client_fd,
             saftbus::Deserializer & received,
             saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

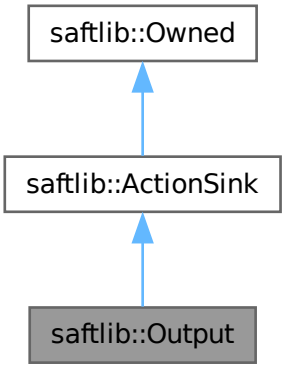
- src/OpenDevice_Service.hpp
- src/OpenDevice_Service.cpp

7.95 saftlib::Output Class Reference

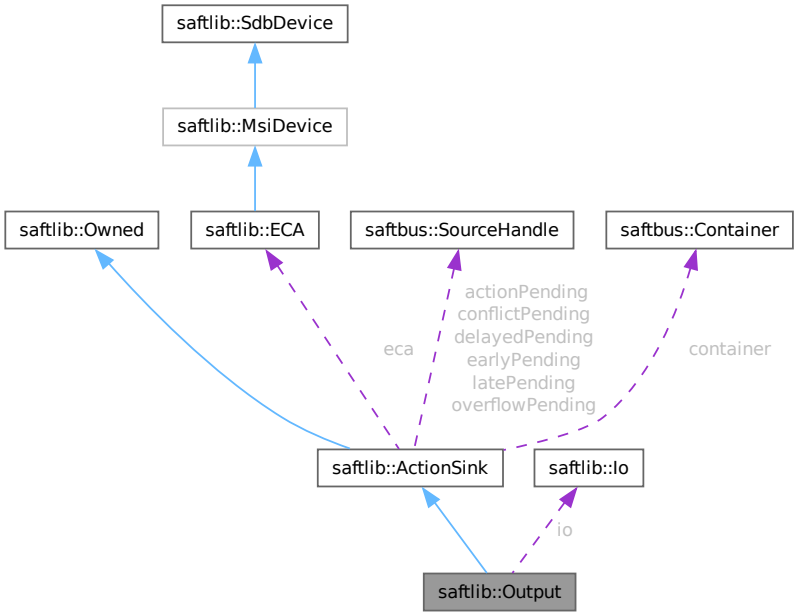
An output through which on/off actions flow.

```
#include <Output.hpp>
```

Inheritance diagram for saftlib::Output:



Collaboration diagram for saftlib::Output:



Public Member Functions

- **Output** (ECA &eca, Io &io, const std::string &output_object_path, const std::string &input_partner_path, unsigned channel, saftbus::Container *container=nullptr)
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, bool on)

  Create a condition to match incoming events.

- void WriteOutput (bool value)

  Directly manipulate the output value.
- bool ReadOutput ()

  Read the output state.
- bool ReadCombinedOutput ()

  Read the combined output state.
- bool StartClock (double high_phase, double low_phase, uint64_t phase_offset)

  Starts the clock generator with given parameters.
- bool ClockStatus (double &high_phase, double &low_phase, uint64_t &phase_offset)

  Get the status of the clock generator.
- bool StopClock ()

  Stops the clock generator.
- uint32_t getIndexOut () const

  IO index.
- void setOutputEnable (bool val)

  Is the output driver enabled.
- bool **getOutputEnable** () const
- void setSpecialPurposeOut (bool val)

  Enable or disable the special function.
- bool **getSpecialPurposeOut** () const
- void setGateOut (bool val)

  Set output gate or get gate status.
- bool **getGateOut** () const
- void setBuTiSMultiplexer (bool val)

  Output BuTiS t0 with timestamp.
- bool **getBuTiSMultiplexer** () const
- void setPPSMultiplexer (bool val)

  Output PPS signal from White Rabbit core.
- bool **getPPSMultiplexer** () const
- bool getOutputEnableAvailable () const

  Can output enable be configured.
- bool getSpecialPurposeOutAvailable () const

  Can special configuration be configured.
- std::string getLogicLevelOut () const

  Logic level of the output.
- std::string getTypeOut () const

  IO type.
- std::string getInput () const

  If non-empty, path of the Input object for the same physical IO.

Public Member Functions inherited from saftlib::ActionSink

- ActionSink (ECA &eca, const std::string &action_sink_object_path, const std::string &name, unsigned channel, unsigned num, saftbus::Container *container=nullptr)

  ActionSink constructor.
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions () const

All conditions created on this ActionSink.

- std::vector< std::string > getActiveConditions () const

  All active conditions created on this ActionSink.

- std::vector< std::string > getInactiveConditions () const

  All inactive conditions created on this ActionSink.

- int64_t getMinOffset () const

  Minimum allowed offset (nanoseconds) usable in NewCondition.

- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset () const

  Maximum allowed offset (nanoseconds) usable in NewCondition.

- void **setMaxOffset** (int64_t val)
- uint64_t getLatency () const

  Nanoseconds between event and earliest execution of an action.

- uint64_t getEarlyThreshold () const

  Actions further into the future than this are early.

- uint16_t getCapacity () const

  The maximum number of actions queueable without Overflow.

- uint16_t getMostFull () const

  Report the largest number of pending actions seen.

- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate () const

  Minimum interval between updates (nanoseconds, default 100ms).

- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount () const

  The number of actions lost due to Overflow.

- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount () const

  The number of actions processed by the Sink.

- void **setActionCount** (uint64_t val)
- uint64_t getLateCount () const

  The number of actions delivered late.

- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount () const

  The number of actions delivered early.

- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount () const

  The number of actions which conflicted.

- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount () const

  The number of actions which have been delayed.

- void **setDelayedCount** (uint64_t val)
- void **compile** ()
- const std::string & **getObjectName** () const
- const std::string & **getObjectPath** () const
- const Conditions & **getConditions** () const
- unsigned **getChannel** () const
- unsigned **getNum** () const
- virtual void **receiveMSI** (uint8_t code)
- Condition * **getCondition** (const std::string object_path)
- void **removeCondition** (Condition *condition)
- unsigned **createConditionNumber** ()
- template<typename ConditionType , typename... Args>
  std::string **NewConditionHelper** (bool active, Args &&... args)

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Protected Attributes

- Io & **io**
- std::string **partnerPath**
- double **clk_low_phase**
- double **clk_high_phase**
- uint64_t **clk_phase_offset**

Protected Attributes inherited from saftlib::ActionSink

- std::string **object_path**
- ECA & **eca**
- std::string **name**
- unsigned **channel**
- unsigned **num**
- int64_t **minOffset**
- int64_t **maxOffset**
- std::chrono::nanoseconds **signalRate**
- uint64_t **overflowCount**
- uint64_t **actionCount**
- uint64_t **lateCount**
- uint64_t **earlyCount**
- uint64_t **conflictCount**
- uint64_t **delayedCount**
- std::chrono::steady_clock::time_point **overflowUpdate**
- std::chrono::steady_clock::time_point **actionUpdate**
- std::chrono::steady_clock::time_point **lateUpdate**
- std::chrono::steady_clock::time_point **earlyUpdate**
- std::chrono::steady_clock::time_point **conflictUpdate**
- std::chrono::steady_clock::time_point **delayedUpdate**
- uint64_t **latency**
- uint64_t **earlyThreshold**

- uint16_t **capacity**
- saftbus::SourceHandle **overflowPending**
- saftbus::SourceHandle **actionPending**
- saftbus::SourceHandle **latePending**
- saftbus::SourceHandle **earlyPending**
- saftbus::SourceHandle **conflictPending**
- saftbus::SourceHandle **delayedPending**
- Conditions **conditions**
- saftbus::Container * **container**

Additional Inherited Members

Public Types inherited from saftlib::ActionSink

- typedef std::map< unsigned, std::unique_ptr< Condition > > **Conditions**

Public Attributes inherited from saftlib::ActionSink

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::ActionSink

- Record **fetchError** (uint8_t code) const
- bool **updateOverflow** () const
- bool **updateAction** () const
- bool **updateLate** () const
- bool **updateEarly** () const
- bool **updateConflict** () const
- bool **updateDelayed** () const

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

7.95.1 Detailed Description

An output through which on/off actions flow.

de.gsi.saftlib.OutputActionSink

This inteface allows the generation of Output pulses. An OutputActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.95.2 Member Function Documentation

7.95.2.1 ClockStatus()

```
bool saftlib::Output::ClockStatus (
            double & high_phase,
            double & low_phase,
            uint64_t & phase_offset )
```

Get the status of the clock generator.

Returns

true if the clock is running

All parameters are in nanoseconds.

7.95.2.2 getIndexOut()

```
uint32_t saftlib::Output::getIndexOut ( ) const
```

IO index.

Returns

IO index

7.95.2.3 getInput()

```
std::string saftlib::Output::getInput ( ) const
```

If non-empty, path of the Input object for the same physical IO.

Returns

object path of the Input object for the same physical IO, or an empty string

7.95.2.4 getLogicLevelOut()

```
std::string saftlib::Output::getLogicLevelOut ( ) const
```

Logic level of the output.

Returns

(LVDS, LVTTL, ...)

7.95.2.5 getOutputEnableAvailable()

```
bool saftlib::Output::getOutputEnableAvailable ( ) const
```

Can output enable be configured.

Returns

true if Output can be enabled or disabled

7.95.2.6 getSpecialPurposeOutAvailable()

```
bool saftlib::Output::getSpecialPurposeOutAvailable ( ) const
```

Can special configuration be configured.

Returns

true if a special purpose function is available

7.95.2.7 getTypeOut()

```
std::string saftlib::Output::getTypeOut ( ) const
```

IO type.

Returns

(GPIO, LVDS, ...)

7.95.2.8 NewCondition()

```
std::string saftlib::Output::NewCondition (
            bool active,
            uint64_t id,
            uint64_t mask,
            int64_t offset,
            bool on )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
on	The output should be toggled on (or off)
result	Object path to the created OutputCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ∼mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a OutputCondition object.

7.95.2.9 ReadCombinedOutput()

```
bool saftlib::Output::ReadCombinedOutput ( )
```

Read the combined output state.

Returns

the combined output value

Every output IO has multiple sources, one or more can be active at the same time. This property shows the real (combined) output state.

7.95.2.10 ReadOutput()

```
bool saftlib::Output::ReadOutput ( )
```

Read the output state.

Returns

true if the output is enabled

This property reflects the current value which would be output when OutputEnable is true. This may differ from ReadInput on an inout.

7.95.2.11 setBuTiSMultiplexer()

```
void saftlib::Output::setBuTiSMultiplexer (
            bool val )
```

Output BuTiS t0 with timestamp.

Parameters

val	true enables BuTiS output

7.95.2.12 setGateOut()

```
void saftlib::Output::setGateOut (
            bool val )
```

Set output gate or get gate status.

Parameters

val	true enables the gate

7.95.2.13 setOutputEnable()

```
void saftlib::Output::setOutputEnable (
            bool val )
```

Is the output driver enabled.

When OutputEnable is false, the output is not driven. This defaults to off. See also Termination if this is an inoutput.

7.95.2.14 setPPSMultiplexer()

```
void saftlib::Output::setPPSMultiplexer (
            bool val )
```

Output PPS signal from White Rabbit core.

Parameters

val	true enables PPS signal

7.95.2.15 setSpecialPurposeOut()

```
void saftlib::Output::setSpecialPurposeOut (
            bool val )
```

Enable or disable the special function.

Parameters

val	true enables the special function

7.95.2.16 StartClock()

```
bool saftlib::Output::StartClock (
            double high_phase,
            double low_phase,
            uint64_t phase_offset )
```

Starts the clock generator with given parameters.

Returns

true if the clock is running

All parameters expect the value in nanoseconds.

7.95.2.17 StopClock()

```
bool saftlib::Output::StopClock ( )
```

Stops the clock generator.

Returns

flase if the clock was stopped

7.95.2.18 WriteOutput()

```
void saftlib::Output::WriteOutput (
            bool value )
```

Directly manipulate the output value.

Parameters

value	true enables the output driver

Set the output to on/off. Overwrite the previous state, regardless of whether it came from WriteOutput or a matching Condition. Similarly, the value may in turn be overwritten by a subsequent matching Condition or WriteOutput.

The documentation for this class was generated from the following files:

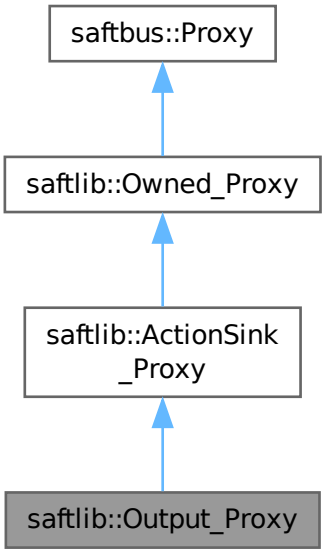
- src/Output.hpp
- src/Output.cpp

7.96 saftlib::Output_Proxy Class Reference

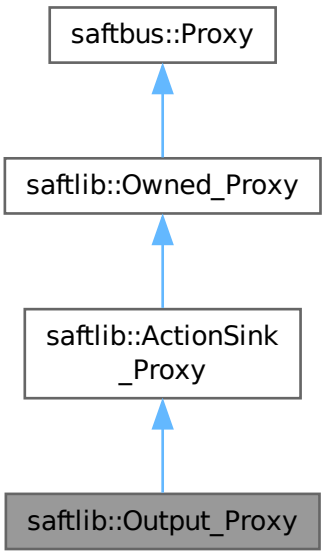
An output through which on/off actions flow.

```
#include <Output_Proxy.hpp>
```

Inheritance diagram for saftlib::Output_Proxy:



Collaboration diagram for saftlib::Output_Proxy:



Public Member Functions

- **Output_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, bool on)

  Create a condition to match incoming events.
- void WriteOutput (bool value)

  Directly manipulate the output value.
- bool ReadOutput ()

  Read the output state.
- bool ReadCombinedOutput ()

  Read the combined output state.
- bool StartClock (double high_phase, double low_phase, uint64_t phase_offset)

  Starts the clock generator with given parameters.
- bool ClockStatus (double &high_phase, double &low_phase, uint64_t &phase_offset)

  Get the status of the clock generator.
- bool StopClock ()

  Stops the clock generator.
- uint32_t getIndexOut ()

  IO index.
- void setOutputEnable (bool val)

  Is the output driver enabled.
- bool **getOutputEnable** ()
- void setSpecialPurposeOut (bool val)

  Enable or disable the special function.
- bool **getSpecialPurposeOut** ()
- void setGateOut (bool val)

  Set output gate or get gate status.
- bool **getGateOut** ()
- void setBuTiSMultiplexer (bool val)

  Output BuTiS t0 with timestamp.
- bool **getBuTiSMultiplexer** ()
- void setPPSMultiplexer (bool val)

  Output PPS signal from White Rabbit core.
- bool **getPPSMultiplexer** ()
- bool getOutputEnableAvailable ()

  Can output enable be configured.
- bool getSpecialPurposeOutAvailable ()

  Can special configuration be configured.
- std::string getLogicLevelOut ()

  Logic level of the output.
- std::string getTypeOut ()

  IO type.
- std::string getInput ()

  If non-empty, path of the Input object for the same physical IO.

Public Member Functions inherited from saftlib::ActionSink_Proxy

- **ActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions ()

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions ()

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions ()

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset ()

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset ()

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency ()

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold ()

  Actions further into the future than this are early.
- uint16_t getCapacity ()

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull ()

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate ()

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount ()

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount ()

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount ()

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount ()

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount ()

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount ()

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< Output_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ActionSink_Proxy

- static std::shared_ptr< ActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::ActionSink_Proxy

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.96.1 Detailed Description

An output through which on/off actions flow.

de.gsi.saftlib.OutputActionSink

This inteface allows the generation of Output pulses. An OutputActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.96.2 Member Function Documentation

7.96.2.1 ClockStatus()

```
bool saftlib::Output_Proxy::ClockStatus (
            double & high_phase,
            double & low_phase,
            uint64_t & phase_offset )
```

Get the status of the clock generator.

Returns

true if the clock is running

All parameters are in nanoseconds.

7.96.2.2 getIndexOut()

```
uint32_t saftlib::Output_Proxy::getIndexOut ( )
```

IO index.

Returns

IO index

7.96.2.3 getInput()

```
std::string saftlib::Output_Proxy::getInput ( )
```

If non-empty, path of the Input object for the same physical IO.

Returns

object path of the Input object for the same physical IO, or an empty string

7.96.2.4 getLogicLevelOut()

```
std::string saftlib::Output_Proxy::getLogicLevelOut ( )
```

Logic level of the output.

Returns

(LVDS, LVTTL, ...)

7.96.2.5 getOutputEnableAvailable()

```
bool saftlib::Output_Proxy::getOutputEnableAvailable ( )
```

Can output enable be configured.

Returns

true if Output can be enabled or disabled

7.96.2.6 getSpecialPurposeOutAvailable()

```
bool saftlib::Output_Proxy::getSpecialPurposeOutAvailable ( )
```

Can special configuration be configured.

Returns

true if a special purpose function is available

7.96.2.7 getTypeOut()

```
std::string saftlib::Output_Proxy::getTypeOut ( )
```

IO type.

Returns

(GPIO, LVDS, ...)

7.96.2.8 NewCondition()

```
std::string saftlib::Output_Proxy::NewCondition (
            bool active,
            uint64_t id,
            uint64_t mask,
            int64_t offset,
            bool on )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
on	The output should be toggled on (or off)
result	Object path to the created OutputCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ∼mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a OutputCondition object.

7.96.2.9 ReadCombinedOutput()

```
bool saftlib::Output_Proxy::ReadCombinedOutput ( )
```

Read the combined output state.

Returns

the combined output value

Every output IO has multiple sources, one or more can be active at the same time. This property shows the real (combined) output state.

7.96.2.10 ReadOutput()

```
bool saftlib::Output_Proxy::ReadOutput ( )
```

Read the output state.

Returns

true if the output is enabled

This property reflects the current value which would be output when OutputEnable is true. This may differ from ReadInput on an inout.

7.96.2.11 setBuTiSMultiplexer()

```
void saftlib::Output_Proxy::setBuTiSMultiplexer (
            bool val )
```

Output BuTiS t0 with timestamp.

Parameters

val	true enables BuTiS output

7.96.2.12 setGateOut()

```
void saftlib::Output_Proxy::setGateOut (
            bool val )
```

Set output gate or get gate status.

Parameters

val	true enables the gate

7.96.2.13 setOutputEnable()

```
void saftlib::Output_Proxy::setOutputEnable (
            bool val )
```

Is the output driver enabled.

When OutputEnable is false, the output is not driven. This defaults to off. See also Termination if this is an inoutput.

7.96.2.14 setPPSMultiplexer()

```
void saftlib::Output_Proxy::setPPSMultiplexer (
            bool val )
```

Output PPS signal from White Rabbit core.

Parameters

val	true enables PPS signal

7.96.2.15 setSpecialPurposeOut()

```
void saftlib::Output_Proxy::setSpecialPurposeOut (
            bool val )
```

Enable or disable the special function.

Parameters

val	true enables the special function

7.96.2.16 signal_dispatch()

```
bool saftlib::Output_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Reimplemented from saftlib::ActionSink_Proxy.

7.96.2.17 StartClock()

```
bool saftlib::Output_Proxy::StartClock (
            double high_phase,
            double low_phase,
            uint64_t phase_offset )
```

Starts the clock generator with given parameters.

Returns

true if the clock is running

All parameters expect the value in nanoseconds.

7.96.2.18 StopClock()

```
bool saftlib::Output_Proxy::StopClock ( )
```

Stops the clock generator.

Returns

flase if the clock was stopped

7.96.2.19 WriteOutput()

```
void saftlib::Output_Proxy::WriteOutput (
            bool value )
```

Directly manipulate the output value.

Parameters

value	true enables the output driver

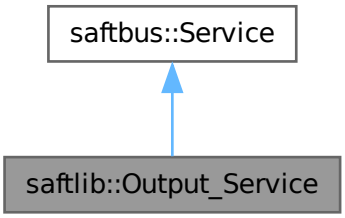
Set the output to on/off. Overwrite the previous state, regardless of whether it came from WriteOutput or a matching Condition. Similarly, the value may in turn be overwritten by a subsequent matching Condition or WriteOutput.

The documentation for this class was generated from the following files:

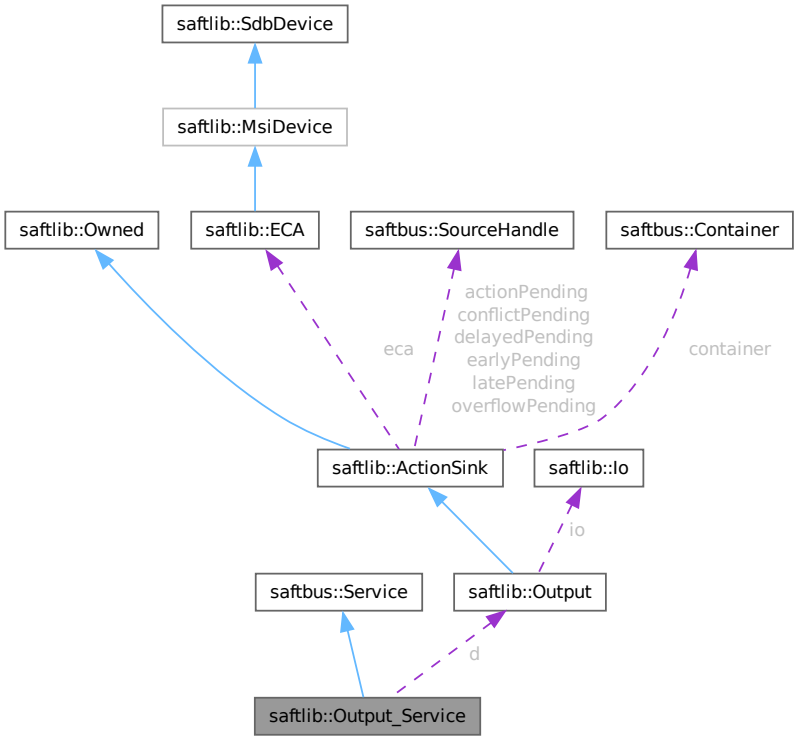
- src/Output_Proxy.hpp
- src/Output_Proxy.cpp

7.97 saftlib::Output_Service Class Reference

Inheritance diagram for saftlib::Output_Service:



Collaboration diagram for saftlib::Output_Service:



Public Types

- typedef Output **DriverType**

Public Member Functions

- **Output_Service** (Output *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **OverflowCount_dispatch_function** (uint64_t arg_1)
- void **ActionCount_dispatch_function** (uint64_t arg_1)
- void **LateCount_dispatch_function** (uint64_t arg_1)
- void **SigLate_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_↩ 4, saftlib::Time arg_5)
- void **EarlyCount_dispatch_function** (uint64_t arg_1)
- void **SigEarly_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_↩ 4, saftlib::Time arg_5)
- void **ConflictCount_dispatch_function** (uint64_t arg_1)
- void **SigConflict_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **DelayedCount_dispatch_function** (uint64_t arg_1)
- void **SigDelayed_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- Output * **d**
- sigc::connection **OverflowCount_connection**
- sigc::connection **ActionCount_connection**
- sigc::connection **LateCount_connection**
- sigc::connection **SigLate_connection**
- sigc::connection **EarlyCount_connection**
- sigc::connection **SigEarly_connection**
- sigc::connection **ConflictCount_connection**
- sigc::connection **SigConflict_connection**
- sigc::connection **DelayedCount_connection**
- sigc::connection **SigDelayed_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.97.1 Member Function Documentation

7.97.1.1 call()

```
void saftlib::Output_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

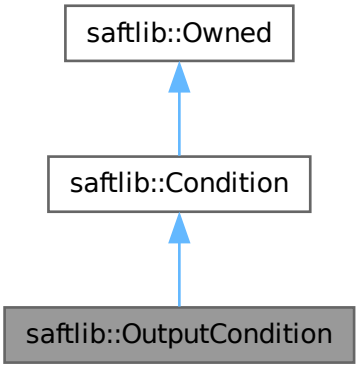
Implements saftbus::Service.

The documentation for this class was generated from the following files:

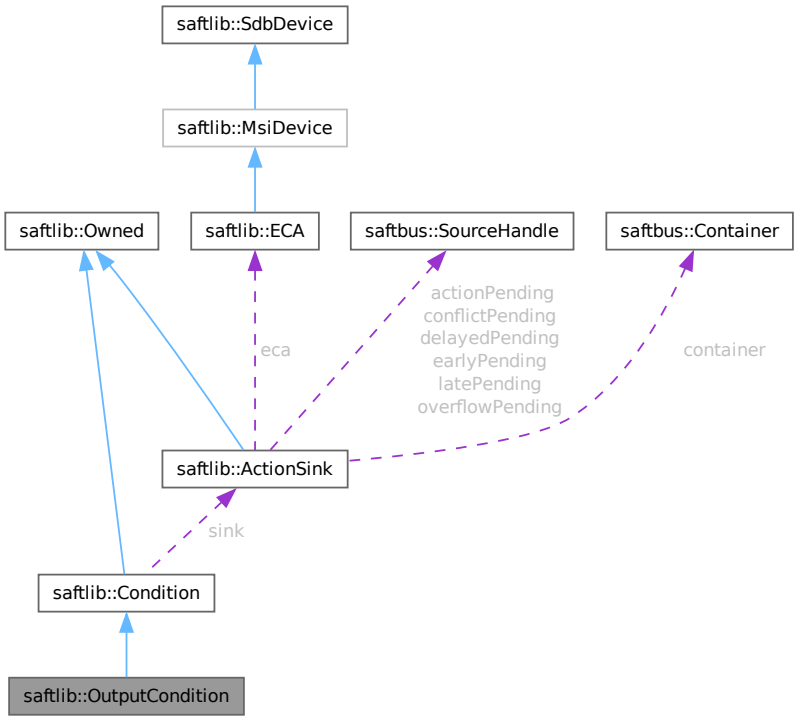
- src/Output_Service.hpp
- src/Output_Service.cpp

7.98 saftlib::OutputCondition Class Reference

Inheritance diagram for saftlib::OutputCondition:



Collaboration diagram for saftlib::OutputCondition:



Public Types

- typedef OutputCondition_Service **ServiceType**

Public Member Functions

- **OutputCondition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container)
- bool getOn () const

  : Matched against incoming events on an OutputActionSink.
- void **setOn** (bool val)

Public Member Functions inherited from saftlib::Condition

- **Condition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)
- uint64_t getID () const

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask () const

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset () const

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate () const

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly () const

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict () const

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed () const

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive () const

  The condition should be actively matched against events.
- void **setActive** (bool val)
- std::string & **getObjectPath** ()
- uint32_t **getRawTag** () const
- void **setRawActive** (bool val)
- unsigned **getNumber** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

Release ownership of the object.

- void Own ()

  Claim ownership of the object.

- std::string getOwner () const

  The client which owns this object.

- bool getDestructible () const

  Can the object be destroyed.

- void Destroy ()

  Destroy this object.

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes inherited from saftlib::Condition

- std::string **objectPath**
- ActionSink * **sink**
- unsigned **number**
- uint64_t **id**
- uint64_t **mask**
- int64_t **offset**
- uint32_t **tag**
- bool **acceptLate**
- bool **acceptEarly**
- bool **acceptConflict**
- bool **acceptDelayed**
- bool **active**

7.98.1 Member Function Documentation

7.98.1.1 getOn()

```
bool saftlib::OutputCondition::getOn ( ) const
```

: Matched against incoming events on an OutputActionSink.

de.gsi.saftlib.OutputCondition:

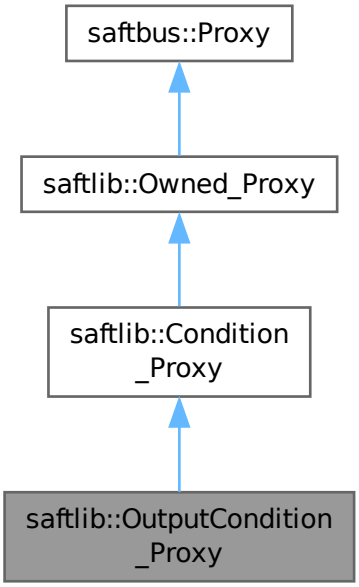
OutputConditions are created by OutputActionSinks to select which events should generate signal toggles. This interface always implies that the object also implements the general Condition interface.

The documentation for this class was generated from the following files:

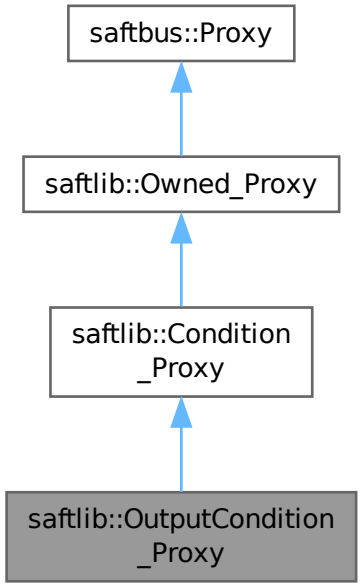
- src/OutputCondition.hpp
- src/OutputCondition.cpp

7.99 saftlib::OutputCondition_Proxy Class Reference

Inheritance diagram for saftlib::OutputCondition_Proxy:



Collaboration diagram for saftlib::OutputCondition_Proxy:



Public Member Functions

- **OutputCondition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::↩ SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- bool getOn ()

  : Matched against incoming events on an OutputActionSink.
- void **setOn** (bool val)

Public Member Functions inherited from saftlib::Condition_Proxy

- **Condition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t getID ()

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask ()

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset ()

Added to an event's time to calculate the action's time.

- void **setOffset** (int64_t val)
- bool getAcceptLate ()

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly ()

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict ()

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed ()

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive ()

  The condition should be actively matched against events.
- void **setActive** (bool val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< OutputCondition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Condition_Proxy

- static std::shared_ptr< Condition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.99.1 Member Function Documentation

7.99.1.1 getOn()

```
bool saftlib::OutputCondition_Proxy::getOn ( )
```

: Matched against incoming events on an OutputActionSink.

de.gsi.saftlib.OutputCondition:

OutputConditions are created by OutputActionSinks to select which events should generate signal toggles. This interface always implies that the object also implements the general Condition interface.

7.99.1.2 signal_dispatch()

```
bool saftlib::OutputCondition_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

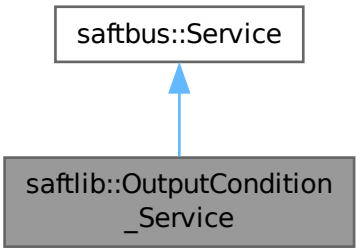
Reimplemented from saftlib::Condition_Proxy.

The documentation for this class was generated from the following files:

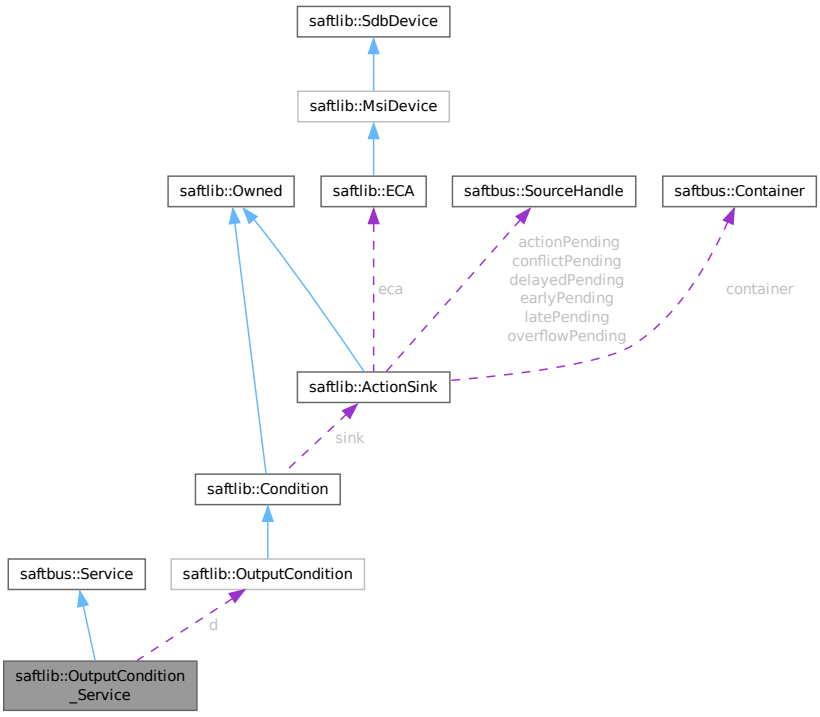
- src/OutputCondition_Proxy.hpp
- src/OutputCondition_Proxy.cpp

7.100 saftlib::OutputCondition_Service Class Reference

Inheritance diagram for saftlib::OutputCondition_Service:



Collaboration diagram for saftlib::OutputCondition_Service:



Public Types

- typedef OutputCondition **DriverType**

Public Member Functions

- **OutputCondition_Service** (OutputCondition *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- OutputCondition * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.100.1 Member Function Documentation

7.100.1.1 call()

```
void saftlib::OutputCondition_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

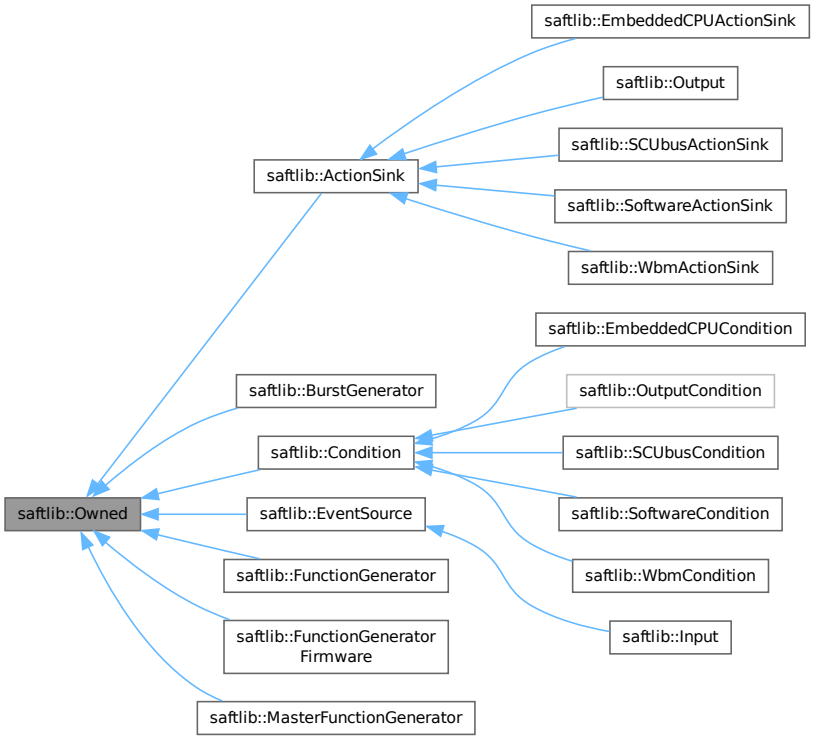
- src/OutputCondition_Service.hpp
- src/OutputCondition_Service.cpp

7.101 saftlib::Owned Class Reference

An object which can grant exclusive access if used in a saftbus::Container.

`#include <Owned.hpp>`

Inheritance diagram for saftlib::Owned:



Public Member Functions

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

Release ownership of the object.

- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Attributes

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

7.101.1 Detailed Description

An object which can grant exclusive access if used in a saftbus::Container.

de.gsi.saftlib.Owned:

This interface allows clients to claim ownership of the object. When the object has no owner, full access is granted to all clients. When owned, only the owner may execute priveledged methods. If the saftbus::Service object has a destruction callback and has an Owner, the object will be automatically Destroyed when the Owner quits.

7.101.2 Member Function Documentation

7.101.2.1 Destroy()

```
void saftlib::Owned::Destroy ( )
```

Destroy this object.

This method may only be invoked by the current owner of the object. However, if the condition has been disowned, it may be invoked by anyone.

7.101.2.2 Disown()

```
void saftlib::Owned::Disown ( )
```

Release ownership of the object.

This method may only be invoked by the current owner of the object. A disowned object may be accessed by all clients and will persist until it is destryed explicitly.

7.101.2.3 getDestructible()

```
bool saftlib::Owned::getDestructible ( ) const
```

Can the object be destroyed.

Returns

true if the object has a destruction_callback registered

A destructible object represents a temporary allocated resource. When the owner quits, the object will be automatically Destoyed. Some objects are indestructible, representing a physical resource.

7.101.2.4 getOwner()

```
std::string saftlib::Owned::getOwner ( ) const
```

The client which owns this object.

Returns

client which owns this object.

If there is no Owner, the empty string is returned. Only the owner may access privileged methods on the object. When the owning client disconnects, ownership will be automatically released, and if the object is Destructable, the object will also be automatically Destroyed.

7.101.2.5 Own()

```
void saftlib::Owned::Own ( )
```

Claim ownership of the object.

This method may only be invoked if the object is unowned.

7.101.3 Member Data Documentation

7.101.3.1 Destroyed

```
sigc::signal<void> saftlib::Owned::Destroyed
```

The object was destroyed.

This signal is only emitted when the function Destroy() is called explicitely. And only if either set_service was called before, or the container in the constructor was set to nullptr.

This signal is not automatically emitted when the Owned destructor is executed.

The documentation for this class was generated from the following files:

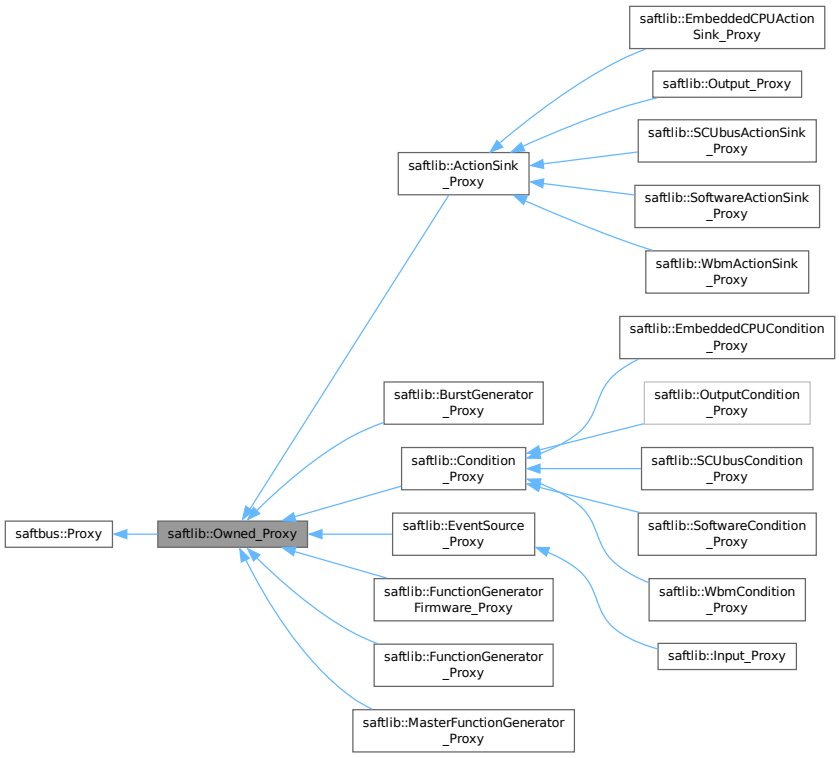
- src/Owned.hpp
- src/Owned.cpp

7.102 saftlib::Owned_Proxy Class Reference

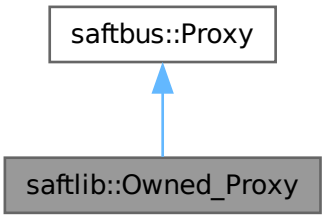
An object which can grant exclusive access if used in a saftbus::Container.

`#include <Owned_Proxy.hpp>`

Inheritance diagram for saftlib::Owned_Proxy:



Collaboration diagram for saftlib::Owned_Proxy:



Public Member Functions

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Public Attributes

- sigc::signal< void > Destroyed

  The object was destroyed.

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.102.1 Detailed Description

An object which can grant exclusive access if used in a saftbus::Container.

de.gsi.saftlib.Owned:

This interface allows clients to claim ownership of the object. When the object has no owner, full access is granted to all clients. When owned, only the owner may execute priveledged methods. If the saftbus::Service object has a destruction callback and has an Owner, the object will be automatically Destroyed when the Owner quits.

7.102.2 Member Function Documentation

7.102.2.1 Destroy()

```
void saftlib::Owned_Proxy::Destroy ( )
```

Destroy this object.

This method may only be invoked by the current owner of the object. However, if the condition has been disowned, it may be invoked by anyone.

7.102.2.2 Disown()

```
void saftlib::Owned_Proxy::Disown ( )
```

Release ownership of the object.

This method may only be invoked by the current owner of the object. A disowned object may be accessed by all clients and will persist until it is destryed explicitly.

7.102.2.3 getDestructible()

```
bool saftlib::Owned_Proxy::getDestructible ( )
```

Can the object be destroyed.

Returns

true if the object has a destruction_callback registered

A destructible object represents a temporary allocated resource. When the owner quits, the object will be automatically Destoyed. Some objects are indestructible, representing a physical resource.

7.102.2.4 getOwner()

```
std::string saftlib::Owned_Proxy::getOwner ( )
```

The client which owns this object.

Returns

client which owns this object.

If there is no Owner, the empty string is returned. Only the owner may access privileged methods on the object. When the owning client disconnects, ownership will be automatically released, and if the object is Destructable, the object will also be automatically Destroyed.

7.102.2.5 Own()

```
void saftlib::Owned_Proxy::Own ( )
```

Claim ownership of the object.

This method may only be invoked if the object is unowned.

7.102.2.6 signal_dispatch()

```
bool saftlib::Owned_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

Reimplemented in saftlib::SCUbusActionSink_Proxy, saftlib::SCUbusCondition_Proxy, saftlib::SoftwareActionSink_Proxy, saftlib::SoftwareCondition_Proxy, saftlib::WbmActionSink_Proxy, and saftlib::WbmCondition_Proxy.

7.102.3 Member Data Documentation

7.102.3.1 Destroyed

```
sigc::signal<void> saftlib::Owned_Proxy::Destroyed
```

The object was destroyed.

This signal is only emitted when the function Destroy() is called explicitely. And only if either set_service was called before, or the container in the constructor was set to nullptr.

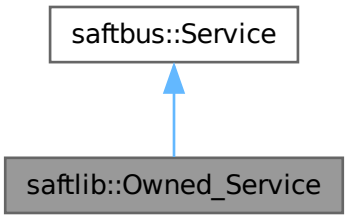
This signal is not automatically emitted when the Owned destructor is executed.

The documentation for this class was generated from the following files:

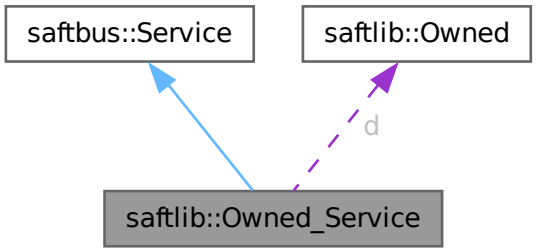
- src/Owned_Proxy.hpp
- src/Owned_Proxy.cpp

7.103 saftlib::Owned_Service Class Reference

Inheritance diagram for saftlib::Owned_Service:



Collaboration diagram for saftlib::Owned_Service:



Public Types

- typedef Owned **DriverType**

Public Member Functions

- **Owned_Service** (Owned *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- Owned * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.103.1 Member Function Documentation

7.103.1.1 call()

```
void saftlib::Owned_Service::call (
             unsigned interface_no,
             unsigned function_no,
             int client_fd,
             saftbus::Deserializer & received,
             saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/Owned_Service.hpp
- src/Owned_Service.cpp

7.104 saftlib::ParameterTuple Struct Reference

Public Member Functions

- uint64_t **duration** () const

Public Attributes

- int16_t **coeff_a**
- int16_t **coeff_b**
- int32_t **coeff_c**
- uint8_t **step**
- uint8_t **freq**

- uint8_t **shift_a**
- uint8_t **shift_b**

The documentation for this struct was generated from the following files:

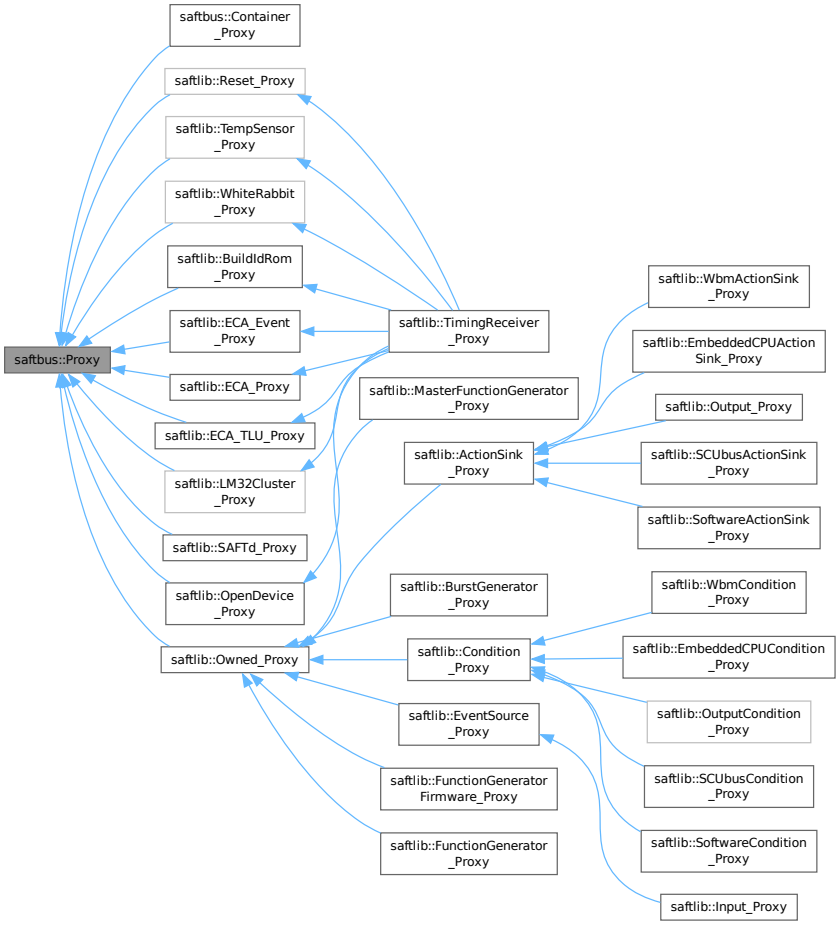
- src/FunctionGeneratorImpl.hpp
- src/FunctionGeneratorImpl.cpp

7.105 saftbus::Proxy Class Reference

Base class of all Proxy objects.

`#include <client.hpp>`

Inheritance diagram for saftbus::Proxy:



Classes

- struct Impl

Public Member Functions

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Protected Member Functions

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

Friends

- class **SignalGroup**

7.105.1 Detailed Description

Base class of all Proxy objects.

Any Proxy object must be derived from saftbus::Proxy. In theory, it is possible to write functional Proxy classes (i.e. derived from this class) By hand, but there are a number of constraints to fulfill for derived functions to work properly. Therefore, Proxy classes are usually generated from a ¨driver class¨ using saftbus-gen, which is part of this software package.

For example, if a class named ¨DriverX¨ is declared in file driverX.hpp and has at least one saftbus tag (// @saftbus-export) on one of its functions or signals, a class DriverX_Service can be generated in file driverX_Proxy.hpp and driverX_Proxy.cpp by calling

```
saftbus-gen driverX.hpp
```

The constructor connects the Proxy with a given Service object (identified by the object_path) and connects the SignalGroup for it. During initialization, the Proxy asks the Service object for the indices that correspond to the given interface names on this particular service object and creates a map. This map available to all Proxy base classes by the method interface_no_from_name.

Parameters

object_path	is the string that identifies the Service object in the saftbus::Container running on the server side.
signal_group	is the SignalGroup over which this Proxy receives its signals.
interfaces_names	is an array of strings with the interface names in text from.

7.105.2 Member Function Documentation

7.105.2.1 get_client_socket_mutex()

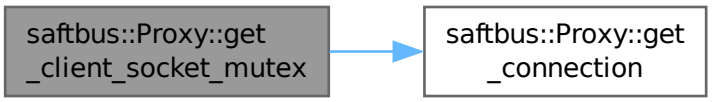
```
std::mutex & saftbus::Proxy::get_client_socket_mutex ( )  [protected]
```

the client socket is a shared resource, it should be locked before using it

Returns

the mutex to lock before using the client socket

Here is the call graph for this function:



7.105.2.2 get_connection()

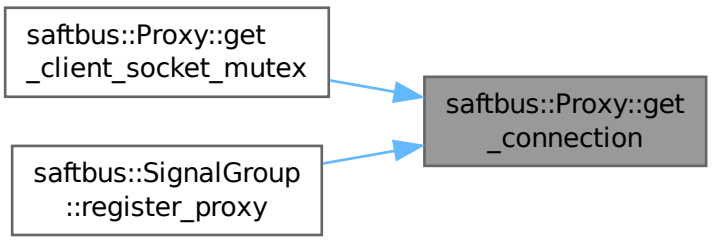
```
ClientConnection & saftbus::Proxy::get_connection ( )  [static], [protected]
```

Get the client connection. Open the connection if that didn't happen before.

Returns

reference to the saftbus::ClientConnection object (there is one per process).

Here is the caller graph for this function:



7.105.2.3 get_proxy_mutex()

```
std::mutex & saftbus::Proxy::get_proxy_mutex ( )  [protected]
```

each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used

Returns

the mutex to lock before using any Proxy resources

7.105.2.4 get_received()

```
Deserializer & saftbus::Proxy::get_received ( )  [protected]
```

Get the deserializer that can be used to receive data from the Service object.

Returns

a reference to saftbus::Deserializer.

7.105.2.5 get_saftbus_object_id()

```
int saftbus::Proxy::get_saftbus_object_id ( )  [protected]
```

The id that was assigned to the Service object of this Proxy.

Returns

the saftbus object id.

7.105.2.6 get_send()

```
Serializer & saftbus::Proxy::get_send ( )  [protected]
```

Get the serializer that can be used to send data to the Service object.

Returns

a reference to saftbus::Serializer.

7.105.2.7 get_signal_group()

```
SignalGroup & saftbus::Proxy::get_signal_group ( )
```

The signal group to which this proxy belongs.

Returns

a reference to a SignalGroup object

7.105.2.8 interface_no_from_name()

```
int saftbus::Proxy::interface_no_from_name (
            const std::string & interface_name )  [protected]
```

needs to be called by derived classes in order to determine which interface_no they refer to.

Proxy only knows the interface name, but not under which number this name can be addressed in the Service object. The Proxy constructor has to get this name->number mapping from the Service object during the initialization phase (the derived Proxy constructor)

7.105.2.9 signal_dispatch()

```
bool saftbus::Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            Deserializer & signal_content )  [pure virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implemented in saftbus::Container_Proxy, saftlib::ActionSink_Proxy, saftlib::BuildIdRom_Proxy, saftlib::BurstGenerator_Proxy, saftlib::Condition_Proxy, saftlib::ECA_Event_Proxy, saftlib::ECA_Proxy, saftlib::ECA_TLU_Proxy, saftlib::EmbeddedCPUActionSink_

saftlib::EmbeddedCPUCondition_Proxy, saftlib::EventSource_Proxy, saftlib::FunctionGenerator_Proxy, saftlib::FunctionGeneratorFirr
saftlib::Input_Proxy, saftlib::LM32Cluster_Proxy, saftlib::MasterFunctionGenerator_Proxy, saftlib::OpenDevice_Proxy,
saftlib::Output_Proxy, saftlib::OutputCondition_Proxy, saftlib::Owned_Proxy, saftlib::Reset_Proxy, saftlib::SAFTd_Proxy,
saftlib::SCUbusActionSink_Proxy, saftlib::SCUbusCondition_Proxy, saftlib::SoftwareActionSink_Proxy, saftlib::SoftwareCondition_Pro
saftlib::TempSensor_Proxy, saftlib::TimingReceiver_Proxy, saftlib::WbmActionSink_Proxy, saftlib::WbmCondition_Proxy,
and saftlib::WhiteRabbit_Proxy.

The documentation for this class was generated from the following files:

- saftbus/client.hpp
- saftbus/client.cpp

7.106 saftlib::ActionSink::Record Struct Reference

Public Attributes

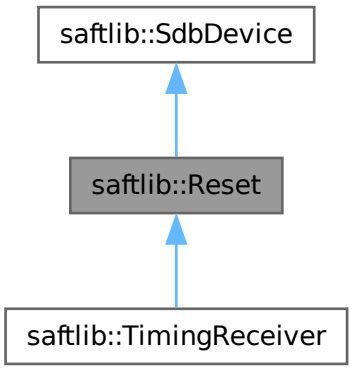
- uint64_t **event**
- uint64_t **param**
- uint64_t **deadline**
- uint64_t **executed**
- uint64_t **count**

The documentation for this struct was generated from the following file:

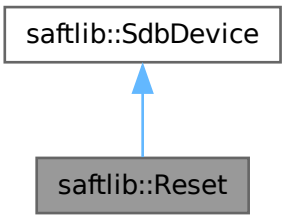
- src/ActionSink.hpp

7.107 saftlib::Reset Class Reference

Inheritance diagram for saftlib::Reset:



Collaboration diagram for saftlib::Reset:



Public Member Functions

- **Reset** (etherbone::Device &device)
- void WdRetrigger ()

  retrigger reset watchdog.
- void CpuHalt (unsigned idx)

  permanently assert reset line of cpu[idx]
- void CpuReset (unsigned idx)

  release reset line of cpu[idx]
- uint32_t CpuHaltStatus ()

  get the 'halt status' of all user lm32 (rightmost bit: CPU 0). bit='1' means halted.

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.107.1 Member Function Documentation

7.107.1.1 CpuHalt()

```
void saftlib::Reset::CpuHalt (
            unsigned idx )
```

permanently assert reset line of cpu[idx]

Parameters

idx	halt cpu[idx] (no check if idx is valid)

7.107.1.2 CpuHaltStatus()

```
uint32_t saftlib::Reset::CpuHaltStatus ( )
```

get the 'halt status' of all user lm32 (rightmost bit: CPU 0). bit='1' means halted.

Returns

32 bits, where bit at position idx represents the halt status of cpu[idx]. '1' means haltet.

7.107.1.3 CpuReset()

```
void saftlib::Reset::CpuReset (
             unsigned idx )
```

release reset line of cpu[idx]

Parameters

idx	cpu[idx] is reset and stats excecuting its program (no check if idx is valid)

7.107.1.4 WdRetrigger()

```
void saftlib::Reset::WdRetrigger ( )
```

retrigger reset watchdog.

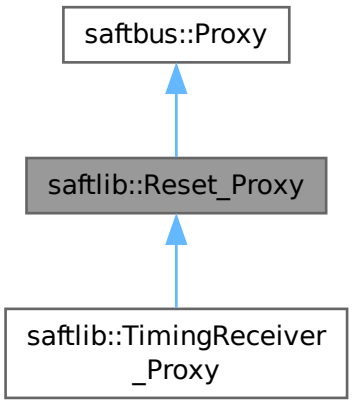
If reset watchdog is enabled and not retriggered within 10 minutes it will reset the FPGA

The documentation for this class was generated from the following files:

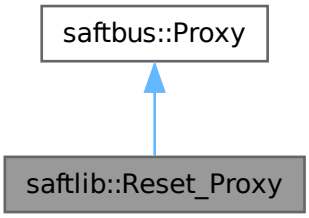
- src/Reset.hpp
- src/Reset.cpp

7.108 saftlib::Reset_Proxy Class Reference

Inheritance diagram for saftlib::Reset_Proxy:



Collaboration diagram for saftlib::Reset_Proxy:



Public Member Functions

- **Reset_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void WdRetrigger ()

  retrigger reset watchdog.
- void CpuHalt (unsigned idx)

  permanently assert reset line of cpu[idx]
- void CpuReset (unsigned idx)

  release reset line of cpu[idx]
- uint32_t CpuHaltStatus ()

  get the 'halt status' of all user lm32 (rightmost bit: CPU 0). bit='1' means halted.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< Reset_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.108.1 Member Function Documentation

7.108.1.1 CpuHalt()

```
void saftlib::Reset_Proxy::CpuHalt (
            unsigned idx )
```

permanently assert reset line of cpu[idx]

Parameters

idx	halt cpu[idx] (no check if idx is valid)

7.108.1.2 CpuHaltStatus()

```
uint32_t saftlib::Reset_Proxy::CpuHaltStatus ( )
```

get the 'halt status' of all user lm32 (rightmost bit: CPU 0). bit='1' means halted.

Returns

32 bits, where bit at position idx represents the halt status of cpu[idx]. '1' means haltet.

7.108.1.3 CpuReset()

```
void saftlib::Reset_Proxy::CpuReset (
            unsigned idx )
```

release reset line of cpu[idx]

Parameters

idx	cpu[idx] is reset and stats excecuting its program (no check if idx is valid)

7.108.1.4 signal_dispatch()

```
bool saftlib::Reset_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

Reimplemented in saftlib::TimingReceiver_Proxy.

7.108.1.5 WdRetrigger()

```
void saftlib::Reset_Proxy::WdRetrigger ( )
```

retrigger reset watchdog.

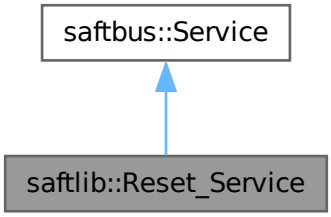
If reset watchdog is enabled and not retriggered within 10 minutes it will reset the FPGA

The documentation for this class was generated from the following files:

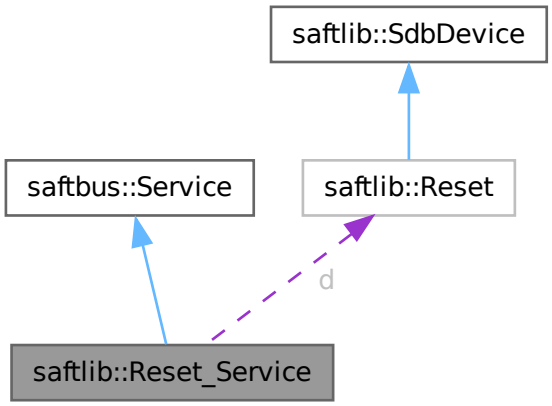
- src/Reset_Proxy.hpp
- src/Reset_Proxy.cpp

7.109 saftlib::Reset_Service Class Reference

Inheritance diagram for saftlib::Reset_Service:



Collaboration diagram for saftlib::Reset_Service:



Public Types

- typedef Reset **DriverType**

Public Member Functions

- **Reset_Service** (Reset *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- Reset * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.109.1 Member Function Documentation

7.109.1.1 call()

```
void saftlib::Reset_Service::call (
             unsigned interface_no,
             unsigned function_no,
             int client_fd,
             saftbus::Deserializer & received,
             saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/Reset_Service.hpp
- src/Reset_Service.cpp

7.110 s_IOCONTROL_SetupField Struct Reference

Public Attributes

- char **uName** [12]
- unsigned char **uSpecial**
- unsigned char **uIndex**
- unsigned char **uIOCfgSpace**
- unsigned char **uLogicLevelRes**

The documentation for this struct was generated from the following file:

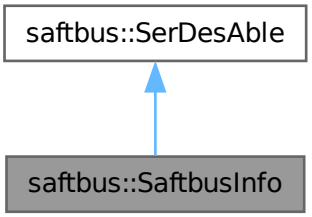
- src/io_control_regs.h

7.111 saftbus::SaftbusInfo Struct Reference

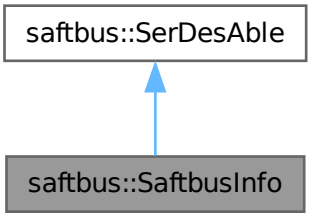
contains all information about the status of a saftbus server.

`#include <client.hpp>`

Inheritance diagram for saftbus::SaftbusInfo:



Collaboration diagram for saftbus::SaftbusInfo:



Classes

- struct ClientInfo

  contains all information about a saftbus client connected to a saftbus server
- struct ObjectInfo

  contains all information about a service object

Public Member Functions

- void serialize (Serializer &ser) const

  custom serializer
- void deserialize (const Deserializer &des)

  custom deserializer

- virtual void serialize (Serializer &ser) const =0
- virtual void deserialize (const Deserializer &des)=0

Public Attributes

- std::vector< ObjectInfo > **object_infos**
- std::vector< ClientInfo > **client_infos**
- std::vector< std::string > **active_plugins**
- std::map< std::string, std::string > **additional_info**

7.111.1 Detailed Description

contains all information about the status of a saftbus server.

7.111.2 Member Function Documentation

7.111.2.1 deserialize()

```
void saftbus::SaftbusInfo::deserialize (
            const Deserializer & des )  [inline], [virtual]
```

custom deserializer

Implements saftbus::SerDesAble.

7.111.2.2 serialize()

```
void saftbus::SaftbusInfo::serialize (
            Serializer & ser ) const  [inline], [virtual]
```

custom serializer

Implements saftbus::SerDesAble.

The documentation for this struct was generated from the following file:

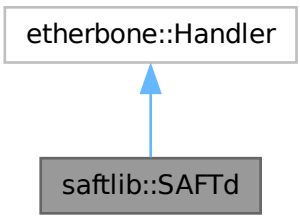
- saftbus/client.hpp

7.112 saftlib::SAFTd Class Reference

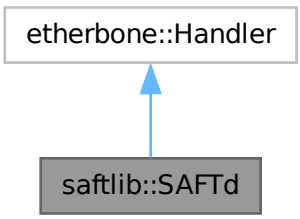
An encapsulated etherbone::Socket with some extra features.

`#include <SAFTd.hpp>`

Inheritance diagram for saftlib::SAFTd:



Collaboration diagram for saftlib::SAFTd:



Public Member Functions

- SAFTd (saftbus::Container *container=nullptr)
  create a new SAFTd instance
- std::string AttachDevice (const std::string &name, const std::string &path, int polling_interval_ms=1)
  Instruct saftd to control a new device.
- void RemoveDevice (const std::string &name)
  Remove the device from saftlib management.
- void Quit ()
  Instructs the saftlib daemon to quit.
- std::string getSourceVersion () const
  SAFTd source version.
- std::string getBuildInfo () const

SAFTd build information.

- std::map< std::string, std::string > getDevices () const

  List of all devices attached to saftd.

- std::string EbForward (const std::string &saftlib_device)

  Get the name of the device that forwards etherbone requests to saftlib_device.

- void release_irq (eb_address_t irq)

  release a callback

- std::unique_ptr< IRQ > request_irq (MsiDevice &msi, const std::function< void(eb_data_t)> &slot)

  register a callback function that can be triggered by an MSI (i.e. a wishbone write access from a master on the MSI crossbar) from the hardware. The wishbone address to trigger the callack is returned from the function. The wishbone data is passed as argument to the callback function.

- std::string getObjectPath ()

  the object path of the SAFTd_Service

- etherbone::Socket & **get_etherbone_socket** ()

  access the underlying ehterbone::Socket

- TimingReceiver * getTimingReceiver (const std::string &object_path)

  access any of the managed TimingReciever driver objects

- eb_status_t read (eb_address_t address, eb_width_t width, eb_data_t *data)

  Implementation of the virtual function etherbone::Handler::read.

- eb_status_t **write** (eb_address_t address, eb_width_t width, eb_data_t data)

  the write function is never used, i.e. Hardware never does read requests towards the host.

7.112.1 Detailed Description

An encapsulated etherbone::Socket with some extra features.

In order to receive message passing interrupts (MSIs) from the Hardware, an instance of SAFTd driver is needed. The name ¨SAFTd¨ is kept for backwards compatibility with older saftlib versions, in order to keep the user facing API stable. A better name would be saftlib::EbSocket, because it encapsulates an etherbone::Socket together with some additional functions. SAFTd provides:

- An instance of an etherbone::Socket with a software eb_slave device connected to it that can receive MSIs When an MSI arrives a connected write function will be called with address and data parameters. The SAFTd class itself is derived from etherbone::Handler and can thus attatch itself to the etherbone::Socket.
- Management of registered callbacks and redistribution of incoming MSIs to registered callback functions. There is a convenient high-level interface to register callbacks from saftlib::MsiDevice instances, which represent wischbone masters on the MSI interconnects on the hardware.
- A container of TimingReceiver objects (std::vector<std::unique_ptr<TimingReceiver> >) and the possibility to add an remove TimingReceiver objects at runtime.

7.112.2 Constructor & Destructor Documentation

7.112.2.1 SAFTd()

```
saftlib::SAFTd::SAFTd (
            saftbus::Container * container = nullptr )
```

create a new SAFTd instance

Parameters

container	if not nullptr, this will be used to register Service objects whenever AttachDevice is called.

7.112.3 Member Function Documentation

7.112.3.1 AttachDevice()

```
std::string saftlib::SAFTd::AttachDevice (
            const std::string & name,
            const std::string & path,
            int polling_interval_ms = 1 )
```

Instruct saftd to control a new device.

Parameters

name	The logical name for the device
path	The etherbone path where the device can be found
polling_interval_ms	Is the MSI polling interval in milliseconds which is only relevant for devices that have no native MSI support

Returns

Object path of the created device

Devices are attached to saftlib by specifying a name and a path. The name should denote the logical relationship of the device to saftd. For example, baseboard would be a good name for the timing receiver attached to an SCU. If an exploder is being used to output events to an oscilloscope, a good logical name might be scope. In these examples, the path for the SCU baseboard would be dev/wbm0, and the scope exploder would be dev/ttyUSB3 or similar. This scheme is intended to make it easy to hot-swap hardware. If the exploder dies, you can simply attach a new one under the same logical name, even though the path might be different.
Here is the call graph for this function:

7.112.3.2 EbForward()

```
std::string saftlib::SAFTd::EbForward (
            const std::string & saftlib_device )
```

Get the name of the device that forwards etherbone requests to saftlib_device.

Parameters

saftlib_device	The name of an attached device.
return	The name any eb-tool can use to communicate with the hardware attached to saftlib_device.

7.112.3.3 getBuildInfo()

```
std::string saftlib::SAFTd::getBuildInfo ( ) const
```

SAFTd build information.

Returns

SAFTd build information.

Information about when and where the SAFTd was compiled. Format is ¨built by USERNAME on MMM DD YYYY HH:MM:SS with HOSTNAME running OPERATING-SYSTEM¨.

7.112.3.4 getDevices()

```
std::map< std::string, std::string > saftlib::SAFTd::getDevices ( ) const
```

List of all devices attached to saftd.

Returns

List of all devices attached to saftd.

The key is the name of the device as provided to AttachDevice. The value is the dbus path to the Device object, NOT the etherbone path of the device. Each object is guaranteed to implement at least the Device interface, however, typically the objects implement the TimingReceiver interface.

7.112.3.5 getObjectPath()

```
std::string saftlib::SAFTd::getObjectPath ( )
```

the object path of the SAFTd_Service

Returns

the object path

7.112.3.6 getSourceVersion()

```
std::string saftlib::SAFTd::getSourceVersion ( ) const
```

SAFTd source version.

The version of the SAFTd source code this daemon was compiled from. Format is ¨saftlib #.#.# (git-id): MMM DD YYYY HH:MM:SS¨.

7.112.3.7 getTimingReceiver()

```
TimingReceiver * saftlib::SAFTd::getTimingReceiver (
            const std::string & object_path )
```

access any of the managed TimingReciever driver objects

Returns

a pointer to the TimingReceiver, thwows if object_path was not found

7.112.3.8 Quit()

```
void saftlib::SAFTd::Quit ( )
```

Instructs the saftlib daemon to quit.

Be absolutely certain before calling this method! All clients will have their future calls throw exceptions.

7.112.3.9 read()

```
eb_status_t saftlib::SAFTd::read (
            eb_address_t address,
            eb_width_t width,
            eb_data_t * data )
```

Implementation of the virtual function etherbone::Handler::read.

read/write virtual functions from etherbone::Handler base class are used to receive incoming etherbone read/write requests from the device. Only write is ever used, an incoming MSI (Message Signal Interrupt) causes a write request.

7.112.3.10 release_irq()

```
void saftlib::SAFTd::release_irq (
            eb_address_t irq )
```

release a callback

Parameters

irq	the address to be released

7.112.3.11 RemoveDevice()

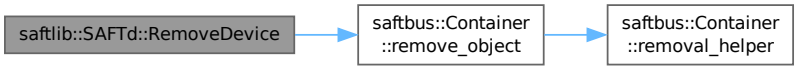
```
void saftlib::SAFTd::RemoveDevice (
            const std::string & name )
```

Remove the device from saftlib management.

Parameters

name	The logical name for the device

Here is the call graph for this function:



7.112.3.12 request_irq()

```
std::unique_ptr< IRQ > saftlib::SAFTd::request_irq (
            MsiDevice & msi,
            const std::function< void(eb_data_t)> & slot )
```

register a callback function that can be triggered by an MSI (i.e. a wishbone write access from a master on the MSI crossbar) from the hardware. The wishbone address to trigger the callack is returned from the function. The wishbone data is passed as argument to the callback function.

Parameters

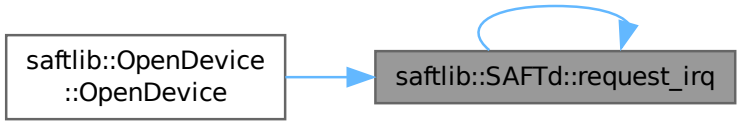
object	of type MsiDevice or derived from MsiDevice. MsiDevices are masters on the MSI-crossbar interconnect
slot	function object that is called when an MSI with the correct address (return value of this fuction) arrives

Returns

an unique_ptr<IRQ> that can be used to obtain the address to trigger the slot function. The irq is released in the destructor of IRQ, i.e. it needs to be stored as long as the interrupt is needed.

Here is the call graph for this function:

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

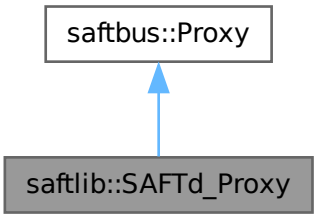
- src/SAFTd.hpp
- src/SAFTd.cpp

7.113 saftlib::SAFTd_Proxy Class Reference

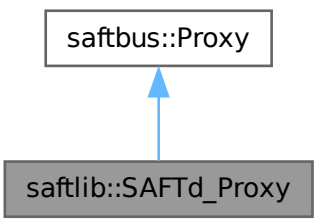
An encapsulated etherbone::Socket with some extra features.

`#include <SAFTd_Proxy.hpp>`

Inheritance diagram for saftlib::SAFTd_Proxy:



Collaboration diagram for saftlib::SAFTd_Proxy:



Public Member Functions

- **SAFTd_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::string AttachDevice (const std::string &name, const std::string &path, int polling_interval_ms=1)

  Instruct saftd to control a new device.
- void RemoveDevice (const std::string &name)

  Remove the device from saftlib management.
- void Quit ()

  Instructs the saftlib daemon to quit.
- std::string getSourceVersion ()

  SAFTd source version.
- std::string getBuildInfo ()

  SAFTd build information.
- std::map< std::string, std::string > getDevices ()

  List of all devices attached to saftd.
- std::string EbForward (const std::string &saftlib_device)

  Get the name of the device that forwards etherbone requests to saftlib_device.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< SAFTd_Proxy > **create** (const std::string &object_path="/de/gsi/saftlib", saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.113.1 Detailed Description

An encapsulated etherbone::Socket with some extra features.

In order to receive message passing interrupts (MSIs) from the Hardware, an instance of SAFTd driver is needed. The name ¨SAFTd¨ is kept for backwards compatibility with older saftlib versions, in order to keep the user facing API stable. A better name would be saftlib::EbSocket, because it encapsulates an etherbone::Socket together with some additional functions. SAFTd provides:

- An instance of an etherbone::Socket with a software eb_slave device connected to it that can receive MSIs When an MSI arrives a connected write function will be called with address and data parameters. The SAFTd class itself is derived from etherbone::Handler and can thus attatch itself to the etherbone::Socket.
- Management of registered callbacks and redistribution of incoming MSIs to registered callback functions. There is a convenient high-level interface to register callbacks from saftlib::MsiDevice instances, which represent wischbone masters on the MSI interconnects on the hardware.
- A container of TimingReceiver objects (std::vector<std::unique_ptr<TimingReceiver> >) and the possibility to add an remove TimingReceiver objects at runtime.

7.113.2 Member Function Documentation

7.113.2.1 AttachDevice()

```
std::string saftlib::SAFTd_Proxy::AttachDevice (
            const std::string & name,
            const std::string & path,
            int polling_interval_ms = 1 )
```

Instruct saftd to control a new device.

Parameters

name	The logical name for the device
path	The etherbone path where the device can be found
polling_interval_ms	Is the MSI polling interval in milliseconds which is only relevant for devices that have no native MSI support

Returns

Object path of the created device

Devices are attached to saftlib by specifying a name and a path. The name should denote the logical relationship of the device to saftd. For example, baseboard would be a good name for the timing receiver attached to an SCU. If an exploder is being used to output events to an oscilloscope, a good logical name might be scope. In these examples, the path for the SCU baseboard would be dev/wbm0, and the scope exploder would be dev/ttyUSB3 or similar. This

scheme is intended to make it easy to hot-swap hardware. If the exploder dies, you can simply attach a new one under the same logical name, even though the path might be different.

7.113.2.2 EbForward()

```
std::string saftlib::SAFTd_Proxy::EbForward (
            const std::string & saftlib_device )
```

Get the name of the device that forwards etherbone requests to saftlib_device.

Parameters

saftlib_device	The name of an attached device.
return	The name any eb-tool can use to communicate with the hardware attached to saftlib_device.

7.113.2.3 getBuildInfo()

```
std::string saftlib::SAFTd_Proxy::getBuildInfo ( )
```

SAFTd build information.

Returns

SAFTd build information.

Information about when and where the SAFTd was compiled. Format is ¨built by USERNAME on MMM DD YYYY HH:MM:SS with HOSTNAME running OPERATING-SYSTEM¨.

7.113.2.4 getDevices()

```
std::map< std::string, std::string > saftlib::SAFTd_Proxy::getDevices ( )
```

List of all devices attached to saftd.

Returns

List of all devices attached to saftd.

The key is the name of the device as provided to AttachDevice. The value is the dbus path to the Device object, NOT the etherbone path of the device. Each object is guaranteed to implement at least the Device interface, however, typically the objects implement the TimingReceiver interface.

7.113.2.5 getSourceVersion()

```
std::string saftlib::SAFTd_Proxy::getSourceVersion ( )
```

SAFTd source version.

The version of the SAFTd source code this daemon was compiled from. Format is ¨saftlib #.#.# (git-id): MMM DD YYYY HH:MM:SS¨.

7.113.2.6 Quit()

```
void saftlib::SAFTd_Proxy::Quit ( )
```

Instructs the saftlib daemon to quit.

Be absolutely certain before calling this method! All clients will have their future calls throw exceptions.

7.113.2.7 RemoveDevice()

```
void saftlib::SAFTd_Proxy::RemoveDevice (
            const std::string & name )
```

Remove the device from saftlib management.

Parameters

name	The logical name for the device

7.113.2.8 signal_dispatch()

```
bool saftlib::SAFTd_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

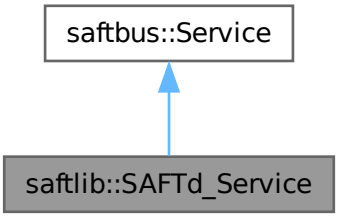
Implements saftbus::Proxy.

The documentation for this class was generated from the following files:

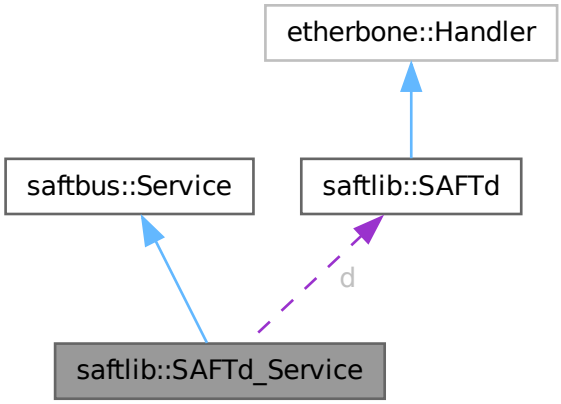
- src/SAFTd_Proxy.hpp
- src/SAFTd_Proxy.cpp

7.114 saftlib::SAFTd_Service Class Reference

Inheritance diagram for saftlib::SAFTd_Service:



Collaboration diagram for saftlib::SAFTd_Service:



Public Types

- typedef SAFTd **DriverType**

Public Member Functions

- **SAFTd_Service** (SAFTd *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- SAFTd * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.114.1 Member Function Documentation

7.114.1.1 call()

```
void saftlib::SAFTd_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

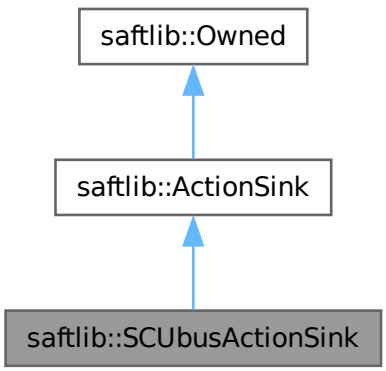
- src/SAFTd_Service.hpp
- src/SAFTd_Service.cpp

7.115 saftlib::SCUbusActionSink Class Reference

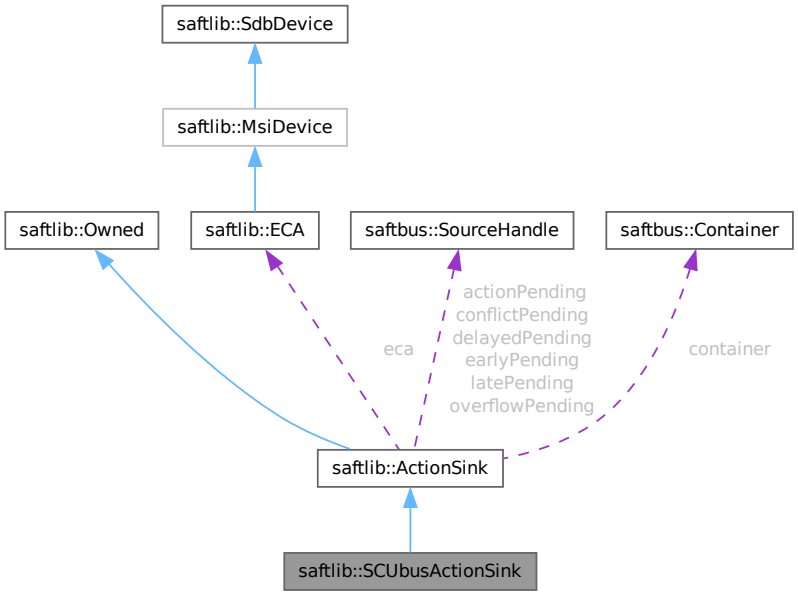
An output through which SCUbus actions flow.

`#include <SCUbusActionSink.hpp>`

Inheritance diagram for saftlib::SCUbusActionSink:



Collaboration diagram for saftlib::SCUbusActionSink:



Public Member Functions

- **SCUbusActionSink** (etherbone::Device &device, ECA &eca, const std::string &object_path, const std::string &name, unsigned channel, eb_address_t scubus_address, saftbus::Container *container=nullptr)
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag)

  Create a condition to match incoming events.
- void InjectTag (uint32_t tag)

  Directly generate a SCUbus timing event.

Public Member Functions inherited from saftlib::ActionSink

- ActionSink (ECA &eca, const std::string &action_sink_object_path, const std::string &name, unsigned channel, unsigned num, saftbus::Container *container=nullptr)

  ActionSink constructor.
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions () const

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions () const

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions () const

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset () const

  Minimum allowed offset (nanoseconds) usable in NewCondition.

- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset () const

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency () const

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold () const

  Actions further into the future than this are early.
- uint16_t getCapacity () const

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull () const

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate () const

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount () const

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount () const

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount () const

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount () const

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount () const

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount () const

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)
- void **compile** ()
- const std::string & **getObjectName** () const
- const std::string & **getObjectPath** () const
- const Conditions & **getConditions** () const
- unsigned **getChannel** () const
- unsigned **getNum** () const
- virtual void **receiveMSI** (uint8_t code)
- Condition * **getCondition** (const std::string object_path)
- void **removeCondition** (Condition *condition)
- unsigned **createConditionNumber** ()
- template<typename ConditionType , typename... Args>
  std::string **NewConditionHelper** (bool active, Args &&... args)

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Protected Attributes

- etherbone::Device & **device**
- eb_address_t **scubus**

Protected Attributes inherited from saftlib::ActionSink

- std::string **object_path**
- ECA & **eca**
- std::string **name**
- unsigned **channel**
- unsigned **num**
- int64_t **minOffset**
- int64_t **maxOffset**
- std::chrono::nanoseconds **signalRate**
- uint64_t **overflowCount**
- uint64_t **actionCount**
- uint64_t **lateCount**
- uint64_t **earlyCount**
- uint64_t **conflictCount**
- uint64_t **delayedCount**
- std::chrono::steady_clock::time_point **overflowUpdate**
- std::chrono::steady_clock::time_point **actionUpdate**
- std::chrono::steady_clock::time_point **lateUpdate**
- std::chrono::steady_clock::time_point **earlyUpdate**
- std::chrono::steady_clock::time_point **conflictUpdate**
- std::chrono::steady_clock::time_point **delayedUpdate**
- uint64_t **latency**
- uint64_t **earlyThreshold**
- uint16_t **capacity**
- saftbus::SourceHandle **overflowPending**
- saftbus::SourceHandle **actionPending**
- saftbus::SourceHandle **latePending**
- saftbus::SourceHandle **earlyPending**
- saftbus::SourceHandle **conflictPending**
- saftbus::SourceHandle **delayedPending**
- Conditions **conditions**
- saftbus::Container * **container**

Additional Inherited Members

Public Types inherited from saftlib::ActionSink

- typedef std::map< unsigned, std::unique_ptr< Condition > > **Conditions**

Public Attributes inherited from saftlib::ActionSink

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::ActionSink

- Record **fetchError** (uint8_t code) const
- bool **updateOverflow** () const
- bool **updateAction** () const
- bool **updateLate** () const
- bool **updateEarly** () const
- bool **updateConflict** () const
- bool **updateDelayed** () const

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

7.115.1 Detailed Description

An output through which SCUbus actions flow.

de.gsi.saftlib.SCUbusActionSink:

This inteface allows the generation of SCU timing events. A SCUbusActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.115.2 Member Function Documentation

7.115.2.1 InjectTag()

```
void saftlib::SCUbusActionSink::InjectTag (
            uint32_t tag )
```

Directly generate a SCUbus timing event.

Parameters

tag	The 32-bit value to push to the SCUbus.

For debugging, it can be helpful to simply create SCUbus events without a matching timing event.

7.115.2.2 NewCondition()

```
std::string saftlib::SCUbusActionSink::NewCondition (
            bool active,
            uint64_t id,
            uint64_t mask,
            int64_t offset,
            uint32_t tag )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
tag	The 32-bit value to send on the SCUbus
result	Object path to the created SCUbusCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ~mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SCUBUSCondition object.

The documentation for this class was generated from the following files:

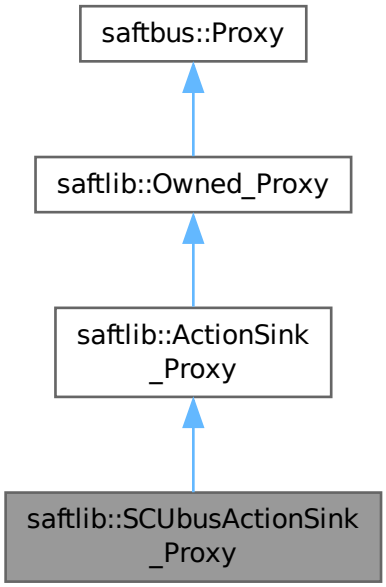
- src/SCUbusActionSink.hpp
- src/SCUbusActionSink.cpp

7.116 saftlib::SCUbusActionSink_Proxy Class Reference

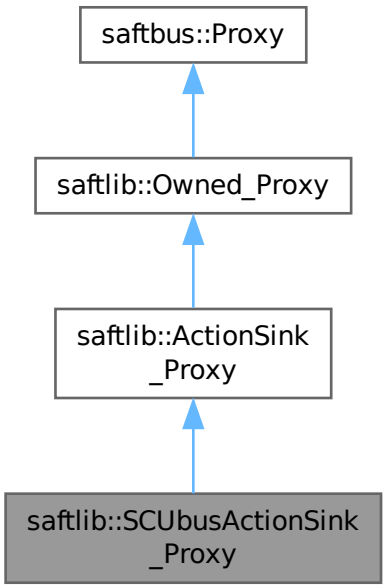
An output through which SCUbus actions flow.

```
#include <SCUbusActionSink_Proxy.hpp>
```

Inheritance diagram for saftlib::SCUbusActionSink_Proxy:



Collaboration diagram for saftlib::SCUbusActionSink_Proxy:



Public Member Functions

- **SCUbusActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus↩ ::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag)

  Create a condition to match incoming events.
- void InjectTag (uint32_t tag)

  Directly generate a SCUbus timing event.

Public Member Functions inherited from saftlib::ActionSink_Proxy

- **ActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions ()

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions ()

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions ()

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset ()

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset ()

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency ()

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold ()

  Actions further into the future than this are early.
- uint16_t getCapacity ()

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull ()

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate ()

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount ()

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount ()

The number of actions processed by the Sink.

- void **setActionCount** (uint64_t val)
- uint64_t getLateCount ()

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount ()

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount ()

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount ()

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< SCUbusActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ActionSink_Proxy

- static std::shared_ptr< ActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::ActionSink_Proxy

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.116.1 Detailed Description

An output through which SCUbus actions flow.

de.gsi.saftlib.SCUbusActionSink:

This inteface allows the generation of SCU timing events. A SCUbusActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.116.2 Member Function Documentation

7.116.2.1 InjectTag()

```
void saftlib::SCUbusActionSink_Proxy::InjectTag (
             uint32_t tag )
```

Directly generate a SCUbus timing event.

Parameters

tag	The 32-bit value to push to the SCUbus.

For debugging, it can be helpful to simply create SCUbus events without a matching timing event.

7.116.2.2 NewCondition()

```
std::string saftlib::SCUbusActionSink_Proxy::NewCondition (
             bool active,
             uint64_t id,
             uint64_t mask,
             int64_t offset,
             uint32_t tag )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
tag	The 32-bit value to send on the SCUbus
result	Object path to the created SCUbusCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ~mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SCUBUSCondition object.

7.116.2.3 signal_dispatch()

```
bool saftlib::SCUbusActionSink_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

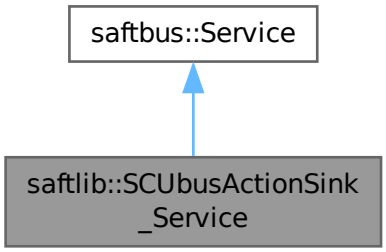
Reimplemented from saftlib::ActionSink_Proxy.

The documentation for this class was generated from the following files:

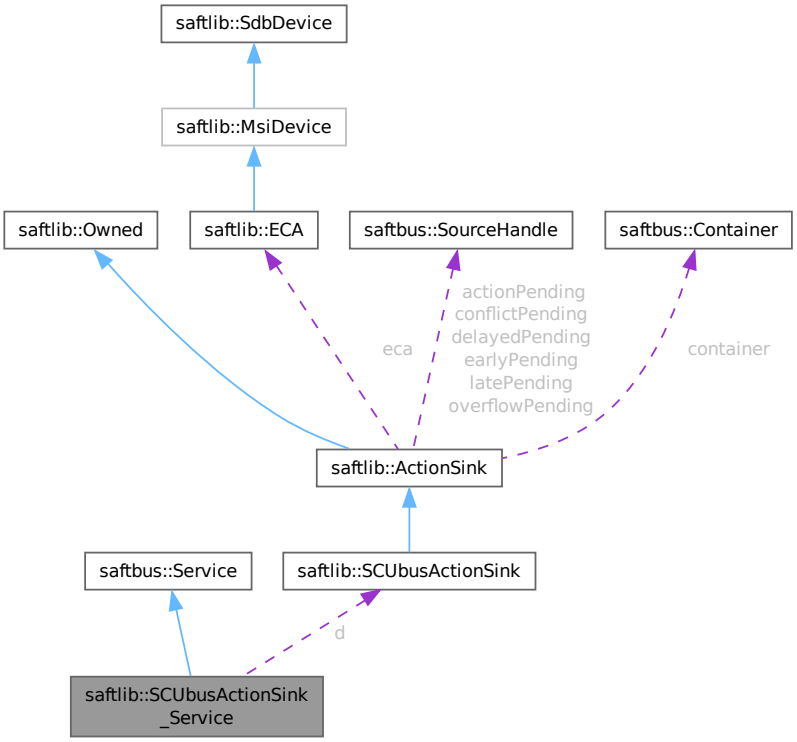
- src/SCUbusActionSink_Proxy.hpp
- src/SCUbusActionSink_Proxy.cpp

7.117 saftlib::SCUbusActionSink_Service Class Reference

Inheritance diagram for saftlib::SCUbusActionSink_Service:



Collaboration diagram for saftlib::SCUbusActionSink_Service:



Public Types

- typedef SCUbusActionSink **DriverType**

Public Member Functions

- **SCUbusActionSink_Service** (SCUbusActionSink *instance, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **OverflowCount_dispatch_function** (uint64_t arg_1)
- void **ActionCount_dispatch_function** (uint64_t arg_1)
- void **LateCount_dispatch_function** (uint64_t arg_1)
- void **SigLate_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_↩ 4, saftlib::Time arg_5)
- void **EarlyCount_dispatch_function** (uint64_t arg_1)
- void **SigEarly_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_↩ 4, saftlib::Time arg_5)
- void **ConflictCount_dispatch_function** (uint64_t arg_1)
- void **SigConflict_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **DelayedCount_dispatch_function** (uint64_t arg_1)
- void **SigDelayed_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- SCUbusActionSink * **d**
- sigc::connection **OverflowCount_connection**
- sigc::connection **ActionCount_connection**
- sigc::connection **LateCount_connection**
- sigc::connection **SigLate_connection**
- sigc::connection **EarlyCount_connection**
- sigc::connection **SigEarly_connection**
- sigc::connection **ConflictCount_connection**
- sigc::connection **SigConflict_connection**
- sigc::connection **DelayedCount_connection**
- sigc::connection **SigDelayed_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.117.1 Member Function Documentation

7.117.1.1 call()

```
void saftlib::SCUbusActionSink_Service::call (
             unsigned interface_no,
             unsigned function_no,
             int client_fd,
             saftbus::Deserializer & received,
             saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

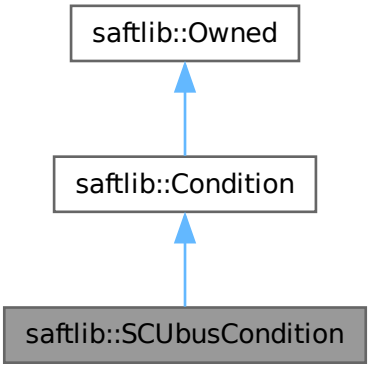
- src/SCUbusActionSink_Service.hpp
- src/SCUbusActionSink_Service.cpp

7.118 saftlib::SCUbusCondition Class Reference

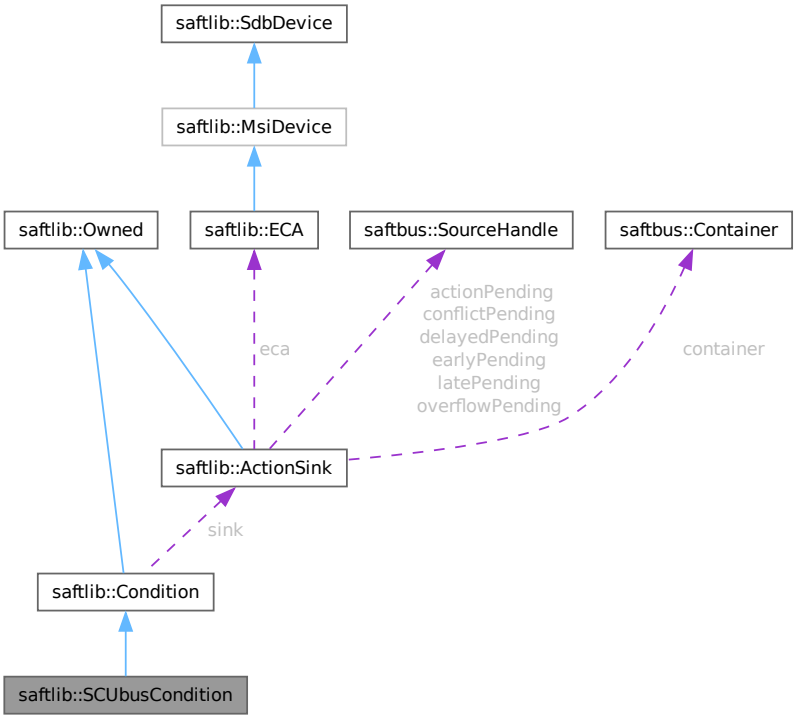
Matched against incoming events on a SCUbusActionSink.

```
#include <SCUbusCondition.hpp>
```

Inheritance diagram for saftlib::SCUbusCondition:



Collaboration diagram for saftlib::SCUbusCondition:



Public Types

- typedef SCUbusCondition_Service **ServiceType**

Public Member Functions

- **SCUbusCondition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)
- uint32_t getTag () const

  The tag which is sent to the SCUbus by this condition.
- void setTag (uint32_t val)

  The tag which is sent to the SCUbus by this condition.

Public Member Functions inherited from saftlib::Condition

- **Condition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)
- uint64_t getID () const

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask () const

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset () const

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate () const

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly () const

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict () const

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed () const

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive () const

  The condition should be actively matched against events.
- void **setActive** (bool val)
- std::string & **getObjectPath** ()
- uint32_t **getRawTag** () const
- void **setRawActive** (bool val)
- unsigned **getNumber** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)

- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes inherited from saftlib::Condition

- std::string **objectPath**
- ActionSink * **sink**
- unsigned **number**
- uint64_t **id**
- uint64_t **mask**
- int64_t **offset**
- uint32_t **tag**
- bool **acceptLate**
- bool **acceptEarly**
- bool **acceptConflict**
- bool **acceptDelayed**
- bool **active**

7.118.1 Detailed Description

Matched against incoming events on a SCUbusActionSink.

de.gsi.saftlib.SCUbusCondition:

SCUbusConditions are created by SCUbusActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

7.118.2 Member Function Documentation

7.118.2.1 getTag()

```
uint32_t saftlib::SCUbusCondition::getTag ( ) const
```

The tag which is sent to the SCUbus by this condition.

Returns

The tag value

7.118.2.2 setTag()

```
void saftlib::SCUbusCondition::setTag (
            uint32_t val )
```

The tag which is sent to the SCUbus by this condition.

Parameters

val	The tag value

The documentation for this class was generated from the following files:

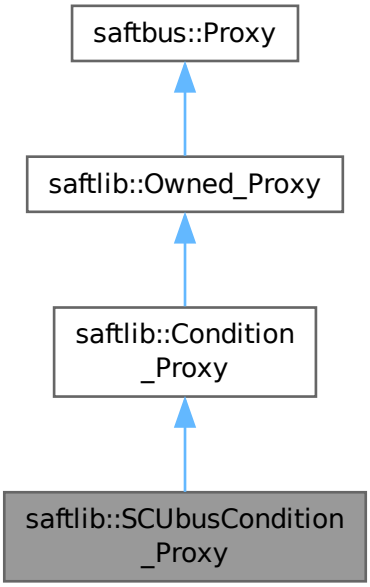
- src/SCUbusCondition.hpp
- src/SCUbusCondition.cpp

7.119 saftlib::SCUbusCondition_Proxy Class Reference

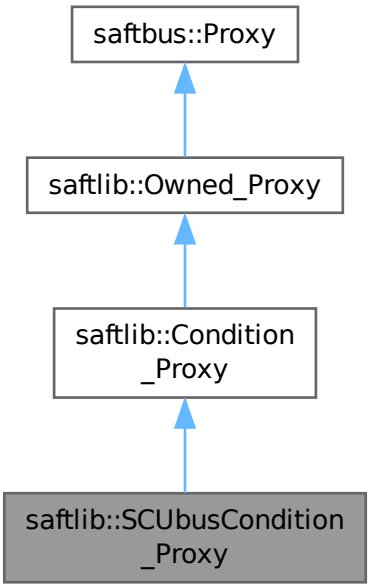
Matched against incoming events on a SCUbusActionSink.

```
#include <SCUbusCondition_Proxy.hpp>
```

Inheritance diagram for saftlib::SCUbusCondition_Proxy:



Collaboration diagram for saftlib::SCUbusCondition_Proxy:



Public Member Functions

- **SCUbusCondition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::↩ SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint32_t getTag ()

  The tag which is sent to the SCUbus by this condition.
- void setTag (uint32_t val)

  The tag which is sent to the SCUbus by this condition.

Public Member Functions inherited from saftlib::Condition_Proxy

- **Condition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t getID ()

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask ()

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset ()

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate ()

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly ()

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict ()

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed ()

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive ()

  The condition should be actively matched against events.
- void **setActive** (bool val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< SCUbusCondition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Condition_Proxy

- static std::shared_ptr< Condition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.119.1 Detailed Description

Matched against incoming events on a SCUbusActionSink.

de.gsi.saftlib.SCUbusCondition:

SCUbusConditions are created by SCUbusActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

7.119.2 Member Function Documentation

7.119.2.1 getTag()

```
uint32_t saftlib::SCUbusCondition_Proxy::getTag ( )
```

The tag which is sent to the SCUbus by this condition.

Returns

The tag value

7.119.2.2 setTag()

```
void saftlib::SCUbusCondition_Proxy::setTag (
            uint32_t val )
```

The tag which is sent to the SCUbus by this condition.

Parameters

val	The tag value

7.119.2.3 signal_dispatch()

```
bool saftlib::SCUbusCondition_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

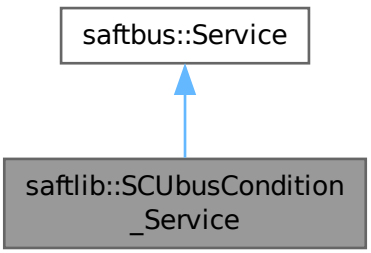
Reimplemented from saftlib::Condition_Proxy.

The documentation for this class was generated from the following files:

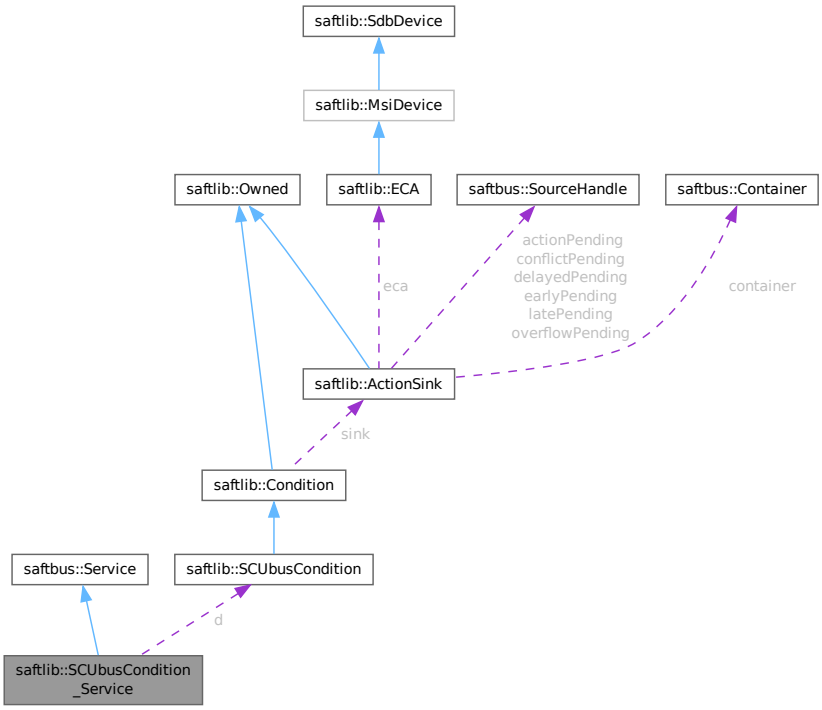
- src/SCUbusCondition_Proxy.hpp
- src/SCUbusCondition_Proxy.cpp

7.120 saftlib::SCUbusCondition_Service Class Reference

Inheritance diagram for saftlib::SCUbusCondition_Service:



Collaboration diagram for saftlib::SCUbusCondition_Service:



Public Types

- typedef SCUbusCondition **DriverType**

Public Member Functions

- **SCUbusCondition_Service** (SCUbusCondition *instance, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- SCUbusCondition * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.120.1 Member Function Documentation

7.120.1.1 call()

```
void saftlib::SCUbusCondition_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

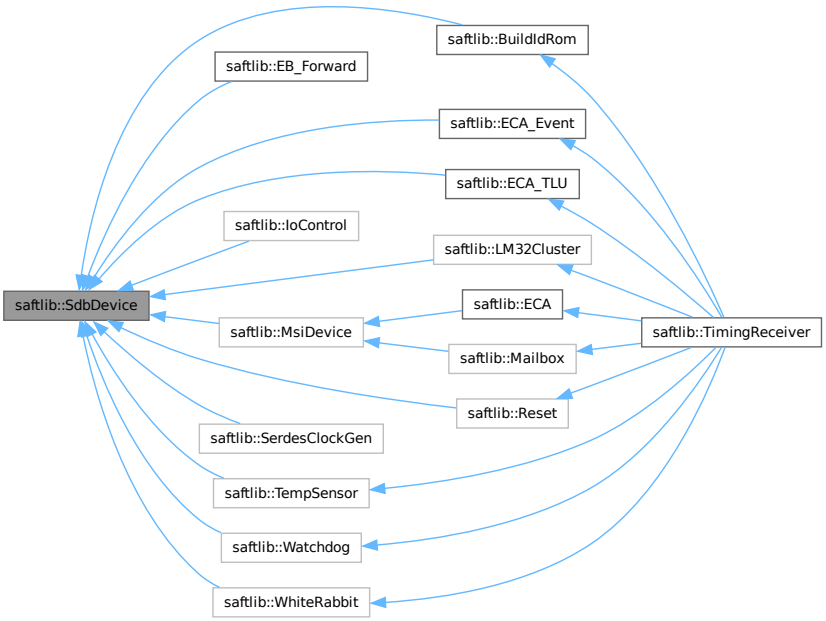
- src/SCUbusCondition_Service.hpp
- src/SCUbusCondition_Service.cpp

7.121 saftlib::SdbDevice Class Reference

SdbDevices calls sdb_find_by_identity and keeps the starting address of the device registers.

`#include <SdbDevice.hpp>`

Inheritance diagram for saftlib::SdbDevice:



Public Member Functions

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Protected Attributes

- eb_address_t **adr_first**
- etherbone::Device & **device**

Friends

- class **SAFTd**

7.121.1 Detailed Description

SdbDevices calls sdb_find_by_identity and keeps the starting address of the device registers.

This class is supposed to be a base class of all driver classes that interact with a single SDB device on the hardware. Deriving from this class avoids to rewrite the boilerplate code to identify a single device. However, it does not support the case when multiple devices with same VENDOR_ID and DEVICE_ID exist. This class always uses the first device listed if more then one exists. It throws an exception if the device is not found at all.

Parameters

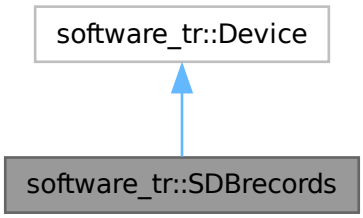
dev	the etherbone::Device
VENDOR_ID	vendor id of the device
DEVICE_ID	device id of the device
throw_if_not_found	in some cases an exception is not desired in which case this parameter can be set to false.

The documentation for this class was generated from the following files:

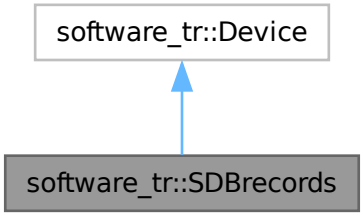
- src/SdbDevice.hpp
- src/SdbDevice.cpp

7.122 software_tr::SDBrecords Class Reference

Inheritance diagram for software_tr::SDBrecords:



Collaboration diagram for software_tr::SDBrecords:



Public Member Functions

- **SDBrecords** (const std::string &filename)
- uint32_t **start_adr** ()
- bool contains (uint32_t adr)
- bool read_access (uint32_t adr, int sel, uint32_t *dat_out)
- template<class Dev >
  std::vector< std::shared_ptr< Dev > > **create_devices** ()
- bool **block_matches_ids** (uint32_t block_adr, uint32_t vendor_id, uint32_t device_id)
- uint32_t **block_device_adr_first** (uint32_t block_adr)

Public Member Functions inherited from software_tr::Device

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.122.1 Member Function Documentation

7.122.1.1 contains()

```
bool software_tr::SDBrecords::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.122.1.2 read_access()

```
bool software_tr::SDBrecords::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

- src/saft-software-tr.cpp

7.123 software_tr::SoftwareECA::Search Struct Reference

Public Member Functions

- **Search** (uint64_t _id, uint64_t _mask, int _first)
- bool **operator==** (const Search &rhs)
- Search & **operator=** (const Search &rhs)

Public Attributes

- uint64_t **id**
- uint64_t **mask**
- int **first**

The documentation for this struct was generated from the following file:

- src/saft-software-tr.cpp

7.124 software_tr::SoftwareECA::SearchCandidate Struct Reference

Public Member Functions

- **SearchCandidate** (int first, uint64_t search)

Public Attributes

- int **first_walker**
- uint64_t **search_event**

The documentation for this struct was generated from the following file:

- src/saft-software-tr.cpp

7.125 saftlib::SearchEntry Struct Reference

Public Member Functions

- **SearchEntry** (uint64_t e, int16_t i)

Public Attributes

- uint64_t **event**
- int16_t **index**

The documentation for this struct was generated from the following file:

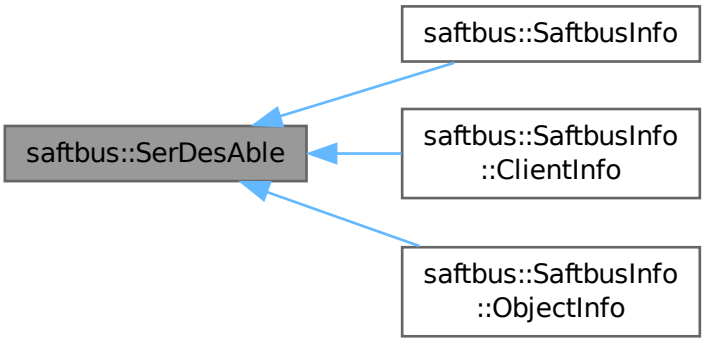
- src/ECA.cpp

7.126 saftbus::SerDesAble Struct Reference

custom types can be sent over saftbus if they derive from this class and implement serialize and deserializ methods

`#include <saftbus.hpp>`

Inheritance diagram for saftbus::SerDesAble:



Public Member Functions

- virtual void serialize (Serializer &ser) const =0
- virtual void deserialize (const Deserializer &des)=0

7.126.1 Detailed Description

custom types can be sent over saftbus if they derive from this class and implement serialize and deserializ methods

7.126.2 Member Function Documentation

7.126.2.1 deserialize()

```
virtual void saftbus::SerDesAble::deserialize (
            const Deserializer & des )  [pure virtual]
```

Implemented in saftbus::SaftbusInfo::ObjectInfo, saftbus::SaftbusInfo::ClientInfo, and saftbus::SaftbusInfo.

7.126.2.2 serialize()

```
virtual void saftbus::SerDesAble::serialize (
            Serializer & ser ) const  [pure virtual]
```

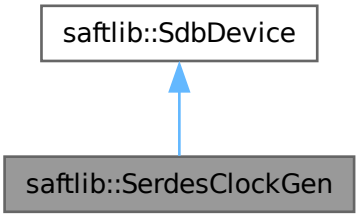
Implemented in saftbus::SaftbusInfo::ObjectInfo, saftbus::SaftbusInfo::ClientInfo, and saftbus::SaftbusInfo.

The documentation for this struct was generated from the following file:

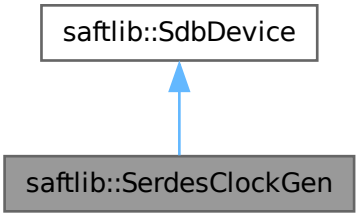
- saftbus/saftbus.hpp

7.127 saftlib::SerdesClockGen Class Reference

Inheritance diagram for saftlib::SerdesClockGen:



Collaboration diagram for saftlib::SerdesClockGen:



Public Member Functions

- **SerdesClockGen** (etherbone::Device &device)
- bool **StartClock** (int io_channel, int io_index, double high_phase, double low_phase, uint64_t phase_offset)
- bool **StopClock** (int io_channel, int io_index)
- bool **ConfigureClock** (int io_channel, int io_index, double high_phase, double low_phase, uint64_t phase↩ _offset)

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

The documentation for this class was generated from the following files:

- src/SerdesClockGen.hpp
- src/SerdesClockGen.cpp

7.128 saftbus::Serializer Class Reference

Simple serializer.

`#include <saftbus.hpp>`

Public Member Functions

- **Serializer** (int reserve=4096)
- bool **write_to** (int fd)
- bool **write_to_no_init** (int fd)
- template<typename T >
  std::enable_if< std::is_base_of< SerDesAble, T >::value >::type **put** (const T &val)
- template<typename T >
  std::enable_if<!std::is_base_of< SerDesAble, T >::value >::type **put** (const T &val)
- template<typename T >
  void **put** (const std::vector< T > &std_vector)
- template<typename T >
  void **put** (const std::vector< std::vector< T, std::allocator< T > >, std::allocator< std::vector< T, std::allocator< T > > > > &std_vector_vector)
- void **put** (const std::string &std_string)
- void **put** (const std::vector< std::string > &vector_string)
- template<typename K , typename V >
  void **put** (const std::map< K, V > &std_map)
- bool **empty** ()
- void **put_init** ()

7.128.1 Detailed Description

Simple serializer.

Classes for serialization and de-serialization do not store type information, i.e. de-serialization only works if the type composition is known (but this is the case in all saftbus transfers) sending data works like this:

- serializer.put(value1);
- serializer.put(value2);
- ...
- serializer.write_to(fd);

receiving data works like this:

- deserializer.read_from(fd);
- deserializer.get(value1);
- deserializer.get(value2);
- ...

The documentation for this class was generated from the following files:

- saftbus/saftbus.hpp
- saftbus/saftbus.cpp

7.129 saftbus::ServerConnection Class Reference

provide a single named UNIX domain socket in the file system and handle client request on that socket

`#include <server.hpp>`

Classes

- struct ClientInfo
- struct Impl

Public Member Functions

- **ServerConnection** (const std::vector< std::pair< std::string, std::vector< std::string > > > &plugins_↩ and_args=std::vector< std::pair< std::string, std::vector< std::string > > >(), const std::string &socket_↩ name="/var/run/saftbus/saftbus")
- void **register_signal_id_for_client** (int client_id, int signal_id)

  Whenever a client sends a signal file descriptor, it should be registered with the Server connection using this function in order to keep track of the use count of signal file descriptors.
- void **unregister_signal_id_for_client** (int client_id, int signal_id)

  Whenever a proxy is unregistered, it should be de-registered with the Server connection using this function in order to keep track of the use count of signal file descriptors.
- int **get_calling_client_id** ()

  return the client id of the currently active client
- Container * **get_container** ()

  access the saftbus::Container that stores all services. The container is owned by the ServerConnection object.
- std::vector< ClientInfo > **get_client_info** ()

7.129.1 Detailed Description

provide a single named UNIX domain socket in the file system and handle client request on that socket

During construction the ServerConnection creates a UNIX domain socket in the file system and listens for incoming connections. If a client connects to that socket, it is expected to send one file descriptor which must be one of the returned elements of the socketpair system call. The ServerConnection maintains this this descriptor until it detects that the client hung up. Since file descriptors are unique integer numbers, they are used to uniquely identify a client.

Parameters

plugins_and_args	Plugins can be directly loaded at startup of the ServerConnection. Each entry in the vector is a filename of a plugin (i.e. a shared object file) and a list of initialization arguments for that plugin. Plugins cann also be loaded later at runtime using the command line tool saftubs-ctl -l <plugin-name> <arg1> <arg2> ... <argn>
socket_name	is the name of the UNIX domain socket which will be opened by the ServerConnection

The documentation for this class was generated from the following files:

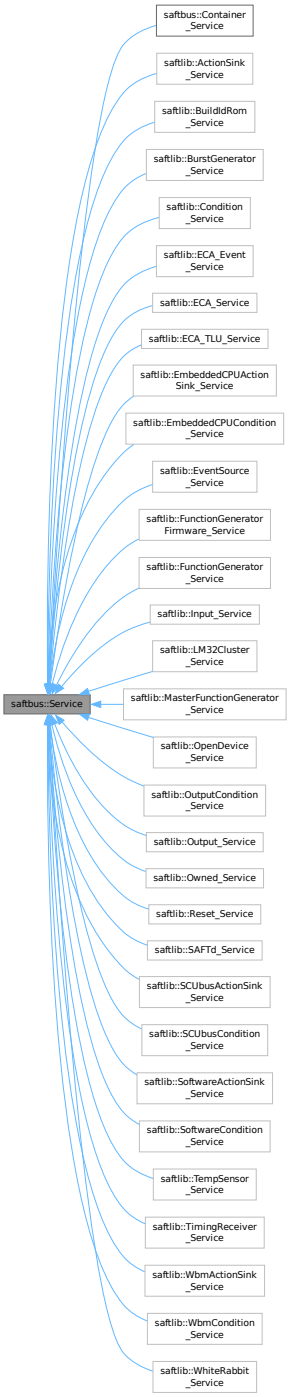
- saftbus/server.hpp
- saftbus/server.cpp

7.130 saftbus::Service Class Reference

base class of all saftbus Services

`#include <service.hpp>`

Inheritance diagram for saftbus::Service:



Classes

- struct Impl

Public Member Functions

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

construct a Service that can be inserted into a saftbus::Container

- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Protected Member Functions

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

Friends

- class **Container**
- class **Container_Service**
- bool **operator==** (std::pair< const unsigned int, std::unique_ptr< saftbus::Service > > &p, const int fd)

7.130.1 Detailed Description

base class of all saftbus Services

Any Service object maintained by saftbus Container must be derived from saftbus::Service. In theory, it is possible to write functional Service classes (i.e. derived from this class) By hand, but there are a number of constraints to fulfill for derived functions to work properly in a saftbus::Container. Therefore, derived classes are usually generated from a ¨driver class¨ using the tool saftbus-gen, which is part of this software package.

For example, if a class named ¨DriverX¨ is declared in file driverX.hpp and has at least one saftbus tag (// @saftbus-export) on one of its functions or signals, a class DriverX_Service can be generated in file driverX_Service.hpp/.cpp by calling

```
saftbus-gen driverX.hpp
```

7.130.2 Constructor & Destructor Documentation

7.130.2.1 Service()

```
saftbus::Service::Service (
            const std::vector< std::string > & interface_names,
            std::function< void()> destruction_callback = std::function<void()>(),
            bool destoy_if_owner_quits = true )
```

construct a Service that can be inserted into a saftbus::Container

Parameters

interface_names	a list of interfaces that are implemented by the Service. Interfaces are usually the name of the driver class and all its base classes.
destruction_callback	is called whenever a Service is removed from saftbus::Container. It allows the driver class to execute some cleanup code in case its related Service object is removed from a saftbus::Container
destroy_if_owner_quits	is true by default, but can be set to false if for some reason the destructible owned service should not be destroyed when its owner quits.

7.130.3 Member Function Documentation

7.130.3.1 call() [1/2]

```
void saftbus::Service::call (
            int client_fd,
            Deserializer & received,
            Serializer & send )
```

execute one of the functions in one of the interfaces of the derived class.

This function extracts interface number and function number from the received data, and calls the pure virtual function call that is implemented in the base class.

Parameters

client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Here is the call graph for this function:

Here is the caller graph for this function:

7.130.3.2 call() [2/2]

```
virtual void saftbus::Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            Deserializer & received,
            Serializer & send )  [protected], [pure virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implemented in saftbus::Container_Service, saftlib::ActionSink_Service, saftlib::BuildIdRom_Service, saftlib::BurstGenerator_Service, saftlib::Condition_Service, saftlib::ECA_Event_Service, saftlib::ECA_Service, saftlib::ECA_TLU_Service, saftlib::EmbeddedCPUActionSink_Service, saftlib::EmbeddedCPUCondition_Service, saftlib::EventSource_Service, saftlib::FunctionGenerator_Service, saftlib::FunctionGeneratorFirmware_Service, saftlib::Input_Service, saftlib::LM32Cluster_Service, saftlib::MasterFunctionGenerator_Service, saftlib::OpenDevice_Service, saftlib::Output_Service, saftlib::OutputCondition_Service, saftlib::Owned_Service, saftlib::Reset_Service, saftlib::SAFTd_Service, saftlib::SCUbusActionSink_Service, saftlib::SCUbusCondition_Service, saftlib::SoftwareActionSink_Service, saftlib::SoftwareCondition_Service, saftlib::TempSensor_Service, saftlib::TimingReceiver_Service, saftlib::WbmActionSink_Service, saftlib::WbmCondition_Service, and saftlib::WhiteRabbit_Service.

7.130.3.3 emit()

```
void saftbus::Service::emit (
            Serializer & send )  [protected]
```

Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).

Parameters

send	the serialized data. The first three values in send must be • the object id (type int) of the Service object in the saftbus::Container • the interface number (type int) of the interface that sends the signal • the signal number (type int) of the signal being sent.

7.130.3.4 get_interface_name2no_map()

```
bool saftbus::Service::get_interface_name2no_map (
            const std::vector< std::string > & interface_names,
            std::map< std::string, int > & interface_name2no_map )
```

obtain a lookup table for the interface names.

The Service class assigns an integer number to each of its interface names. The number is used in remote function calls to address the correct interface. This function generates a lookup table from name to assigned number. Proxy objects of this service use this number to refer to an interface when executing a remote function call over saftbus. This function is meant to be called by derived Classes generated with saftbus-gen.

Parameters

interface_names	a list of all interface for which the assigned number is needed.
interface_name2no_map	the lookup table with the correct integer number assigned to each of the names in interface_names.

Returns

true if all requested interface names are implemented by the Service calss, false otherwise.

The documentation for this class was generated from the following files:

- saftbus/service.hpp
- saftbus/service.cpp

7.131 saftbus::SignalGroup Class Reference

Manage incoming saftbus signals and distribute them to the connected Proxy objects.

```
#include <client.hpp>
```

Classes

- struct Impl

Public Member Functions

- int **register_proxy** (Proxy *proxy)

  used in the Constructor of Proxy objects to connect themselves to this SignalGroup.
- void **unregister_proxy** (Proxy *proxy)

  used in the Destructor of Proxy objects to remove themselves from this SignalGroup.
- int get_fd ()

  Get the file descriptor where the signal are being sent.
- int wait_for_signal (int timeout_ms=-1)

  Wait for signal to arrive and return either on timeout, or when a number of signals was dispatcht and there are no more signals in the pipe.
- int wait_for_one_signal (int timeout_ms=-1)

  Wait for a signal to arrive and return either on timeout, or when one signal was dispatched.

Static Public Member Functions

- static SignalGroup & **get_global** ()

Friends

- class **Proxy**

7.131.1 Detailed Description

Manage incoming saftbus signals and distribute them to the connected Proxy objects.

Signals from Services are always sent to an instance of SignalGroup, which manages a file descriptor, and can wait for incoming signals on this file descriptor (wait_for_signal). If signals arrive, they are distributes to all registered Proxy objects with a matching interface (to which the signal is addressed). By default, proxy objects are connected to the saftbus::SignalGroup::get_global() signal group. In some situations (e.g. when independent threads are used), Proxy objects might need their own channel for signals. A new SignalGroup can be created and passed to the constructor of Proxy objects in order to assign them to this SignalGroup.

7.131.2 Member Function Documentation

7.131.2.1 get_fd()

```
int saftbus::SignalGroup::get_fd ( )
```

Get the file descriptor where the signal are being sent.

This function is intended to be used when saftbus signals need to be integrated into an event loop.

7.131.2.2 wait_for_one_signal()

```
int saftbus::SignalGroup::wait_for_one_signal (
            int timeout_ms = -1 )
```

Wait for a signal to arrive and return either on timeout, or when one signal was dispatched.

Use this function only if exactly one signal is to be dispatched. Otherwise use wait_for_signal.

Parameters

timeout_ms	Don't wait longer than so many milliseconds.

Returns

>0 if a signal was received, 0 if timeout was hit, < 0 in case of failure (e.g. service object was destroyed)

7.131.2.3 wait_for_signal()

```
int saftbus::SignalGroup::wait_for_signal (
            int timeout_ms = -1 )
```

Wait for signal to arrive and return either on timeout, or when a number of signals was dispatcht and there are no more signals in the pipe.

In most situations, this function should be preferred over wait_for_one_signal.

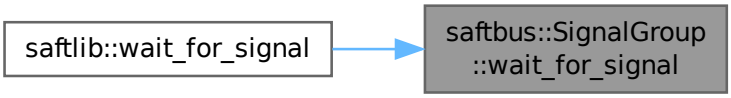
Parameters

timeout_ms	Don't wait longer than so many milliseconds.

Returns

>0 if a signal was received, 0 if timeout was hit, < 0 in case of failure (e.g. service object was destroyed)

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- saftbus/client.hpp
- saftbus/client.cpp

7.132 saftlib::Mailbox::Slot Class Reference

Public Member Functions

- ~**Slot** ()

free Mailbox hardware ressources for that slot

- int getIndex ()

  the slot index that is owned

- eb_address_t getAddress ()

  address of the owed slot

- void Use (uint32_t value)

  write a value to the preconfigured address.

Friends

- class **Mailbox**

7.132.1 Member Function Documentation

7.132.1.1 getAddress()

```
eb_address_t saftlib::Mailbox::Slot::getAddress ( )
```

address of the owed slot

Returns

write to this eb-address to use the slot

7.132.1.2 getIndex()

```
int saftlib::Mailbox::Slot::getIndex ( )
```

the slot index that is owned

Returns

slot index

7.132.1.3 Use()

```
void saftlib::Mailbox::Slot::Use (
            uint32_t value )
```

write a value to the preconfigured address.

Parameters

slot_index	the slot to be used (return value of ConfigureSlot)
value	is the value to be written

The documentation for this class was generated from the following files:

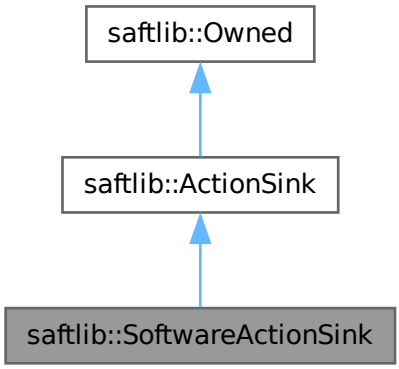
- src/Mailbox.hpp
- src/Mailbox.cpp

7.133 saftlib::SoftwareActionSink Class Reference

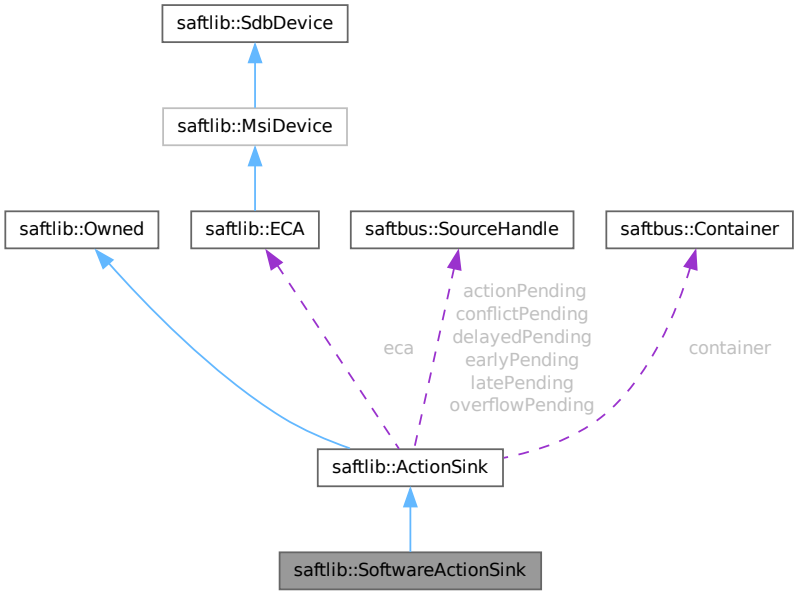
An output through which software actions flow.

`#include <SoftwareActionSink.hpp>`

Inheritance diagram for saftlib::SoftwareActionSink:



Collaboration diagram for saftlib::SoftwareActionSink:



Public Member Functions

- **SoftwareActionSink** (ECA &eca, const std::string &object_path, const std::string &name, unsigned channel, unsigned num, eb_address_t queue_address, saftbus::Container *container=nullptr)
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset)
- void receiveMSI (uint8_t code)
- SoftwareCondition * **getCondition** (const std::string object_path)

Public Member Functions inherited from saftlib::ActionSink

- ActionSink (ECA &eca, const std::string &action_sink_object_path, const std::string &name, unsigned channel, unsigned num, saftbus::Container *container=nullptr)

  ActionSink constructor.
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions () const

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions () const

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions () const

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset () const

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset () const

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency () const

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold () const

  Actions further into the future than this are early.
- uint16_t getCapacity () const

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull () const

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate () const

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount () const

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount () const

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount () const

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount () const

The number of actions delivered early.

- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount () const

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount () const

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)
- void **compile** ()
- const std::string & **getObjectName** () const
- const std::string & **getObjectPath** () const
- const Conditions & **getConditions** () const
- unsigned **getChannel** () const
- unsigned **getNum** () const
- virtual void **receiveMSI** (uint8_t code)
- Condition * **getCondition** (const std::string object_path)
- void **removeCondition** (Condition *condition)
- unsigned **createConditionNumber** ()
- template<typename ConditionType , typename... Args>
  std::string **NewConditionHelper** (bool active, Args &&... args)

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Protected Attributes

- eb_address_t **queue**

Protected Attributes inherited from saftlib::ActionSink

- std::string **object_path**
- ECA & **eca**
- std::string **name**
- unsigned **channel**
- unsigned **num**
- int64_t **minOffset**
- int64_t **maxOffset**
- std::chrono::nanoseconds **signalRate**
- uint64_t **overflowCount**
- uint64_t **actionCount**
- uint64_t **lateCount**
- uint64_t **earlyCount**
- uint64_t **conflictCount**
- uint64_t **delayedCount**
- std::chrono::steady_clock::time_point **overflowUpdate**
- std::chrono::steady_clock::time_point **actionUpdate**
- std::chrono::steady_clock::time_point **lateUpdate**
- std::chrono::steady_clock::time_point **earlyUpdate**
- std::chrono::steady_clock::time_point **conflictUpdate**
- std::chrono::steady_clock::time_point **delayedUpdate**
- uint64_t **latency**
- uint64_t **earlyThreshold**
- uint16_t **capacity**
- saftbus::SourceHandle **overflowPending**
- saftbus::SourceHandle **actionPending**
- saftbus::SourceHandle **latePending**
- saftbus::SourceHandle **earlyPending**
- saftbus::SourceHandle **conflictPending**
- saftbus::SourceHandle **delayedPending**
- Conditions **conditions**
- saftbus::Container * **container**

Additional Inherited Members

Public Types inherited from saftlib::ActionSink

- typedef std::map< unsigned, std::unique_ptr< Condition > > **Conditions**

Public Attributes inherited from saftlib::ActionSink

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::ActionSink

- Record **fetchError** (uint8_t code) const
- bool **updateOverflow** () const
- bool **updateAction** () const
- bool **updateLate** () const
- bool **updateEarly** () const
- bool **updateConflict** () const
- bool **updateDelayed** () const

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

7.133.1 Detailed Description

An output through which software actions flow.

de.gsi.saftlib.SoftwareActionSink:

A SoftwareActionSink guarantees ordered execution of all callbacks on SoftwareConditions created via the NewCondition method. Each SoftwareActionSink is independent of all others, so a single program may operate in isolation without concern about potential conflicting rules from other clients on the same machine.

A SoftwareActionSink is both an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.133.2 Member Function Documentation

7.133.2.1 NewCondition()

```
std::string saftlib::SoftwareActionSink::NewCondition (
            bool active,
            uint64_t id,
            uint64_t mask,
            int64_t offset )
```

NewCondition: Create a condition to match incoming events

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action

Returns

Object path to the created SoftwareCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ~mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SoftwareCondition object.

7.133.2.2 receiveMSI()

```
void saftlib::SoftwareActionSink::receiveMSI (
            uint8_t code )  [virtual]
```

Reimplemented from saftlib::ActionSink.

The documentation for this class was generated from the following files:

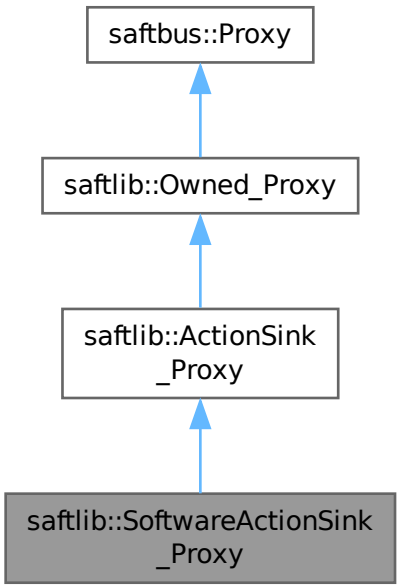
- src/SoftwareActionSink.hpp
- src/SoftwareActionSink.cpp

7.134 saftlib::SoftwareActionSink_Proxy Class Reference

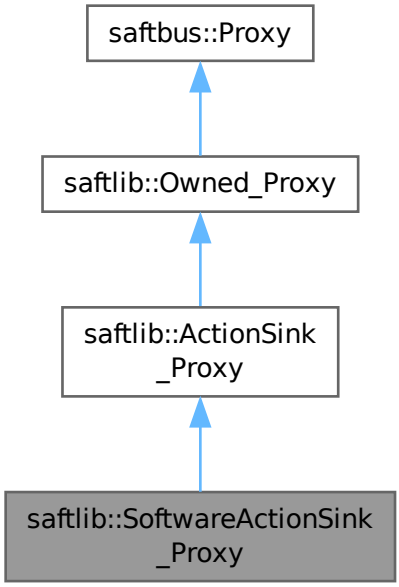
An output through which software actions flow.

`#include <SoftwareActionSink_Proxy.hpp>`

Inheritance diagram for saftlib::SoftwareActionSink_Proxy:



Collaboration diagram for saftlib::SoftwareActionSink_Proxy:



Public Member Functions

- **SoftwareActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus↩ ::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset)

Public Member Functions inherited from saftlib::ActionSink_Proxy

- **ActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions ()

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions ()

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions ()

All inactive conditions created on this ActionSink.

- int64_t getMinOffset ()

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset ()

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency ()

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold ()

  Actions further into the future than this are early.
- uint16_t getCapacity ()

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull ()

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate ()

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount ()

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount ()

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount ()

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount ()

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount ()

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount ()

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< SoftwareActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ActionSink_Proxy

- static std::shared_ptr< ActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::ActionSink_Proxy

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.134.1 Detailed Description

An output through which software actions flow.

de.gsi.saftlib.SoftwareActionSink:

A SoftwareActionSink guarantees ordered execution of all callbacks on SoftwareConditions created via the NewCondition method. Each SoftwareActionSink is independent of all others, so a single program may operate in isolation without concern about potential conflicting rules from other clients on the same machine.

A SoftwareActionSink is both an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.134.2 Member Function Documentation

7.134.2.1 NewCondition()

```
std::string saftlib::SoftwareActionSink_Proxy::NewCondition (
            bool active,
            uint64_t id,
            uint64_t mask,
            int64_t offset )
```

NewCondition: Create a condition to match incoming events

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action

Returns

Object path to the created SoftwareCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ∼mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SoftwareCondition object.

7.134.2.2 signal_dispatch()

```
bool saftlib::SoftwareActionSink_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

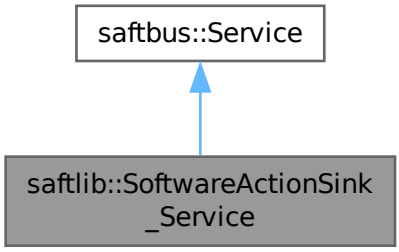
Reimplemented from saftlib::ActionSink_Proxy.

The documentation for this class was generated from the following files:

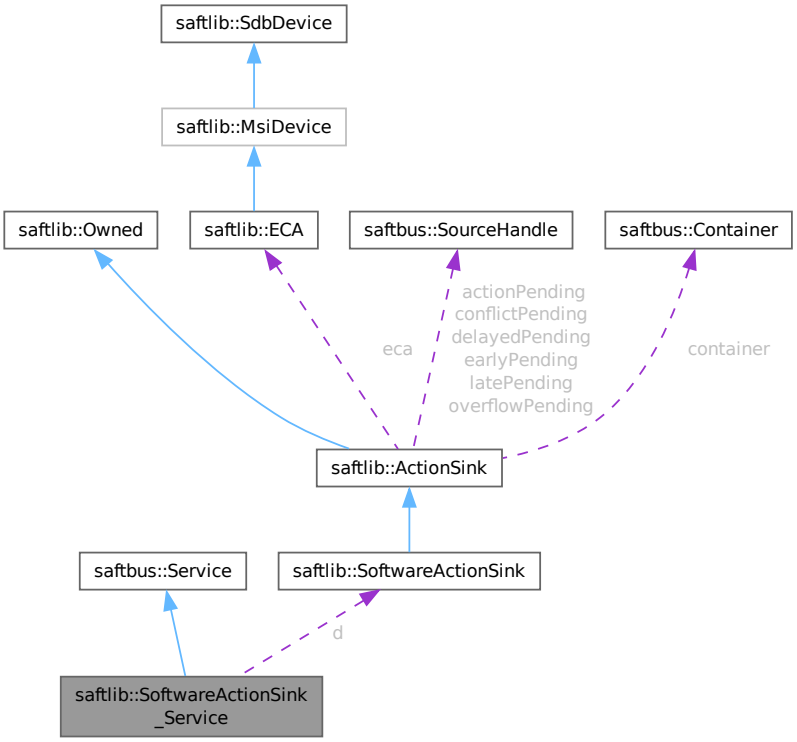
- src/SoftwareActionSink_Proxy.hpp
- src/SoftwareActionSink_Proxy.cpp

7.135 saftlib::SoftwareActionSink_Service Class Reference

Inheritance diagram for saftlib::SoftwareActionSink_Service:



Collaboration diagram for saftlib::SoftwareActionSink_Service:



Public Types

- typedef SoftwareActionSink **DriverType**

Public Member Functions

- **SoftwareActionSink_Service** (SoftwareActionSink *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **OverflowCount_dispatch_function** (uint64_t arg_1)
- void **ActionCount_dispatch_function** (uint64_t arg_1)
- void **LateCount_dispatch_function** (uint64_t arg_1)
- void **SigLate_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **EarlyCount_dispatch_function** (uint64_t arg_1)
- void **SigEarly_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **ConflictCount_dispatch_function** (uint64_t arg_1)
- void **SigConflict_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **DelayedCount_dispatch_function** (uint64_t arg_1)
- void **SigDelayed_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- SoftwareActionSink * **d**
- sigc::connection **OverflowCount_connection**
- sigc::connection **ActionCount_connection**
- sigc::connection **LateCount_connection**
- sigc::connection **SigLate_connection**
- sigc::connection **EarlyCount_connection**
- sigc::connection **SigEarly_connection**
- sigc::connection **ConflictCount_connection**
- sigc::connection **SigConflict_connection**
- sigc::connection **DelayedCount_connection**
- sigc::connection **SigDelayed_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.135.1 Member Function Documentation

7.135.1.1 call()

```
void saftlib::SoftwareActionSink_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

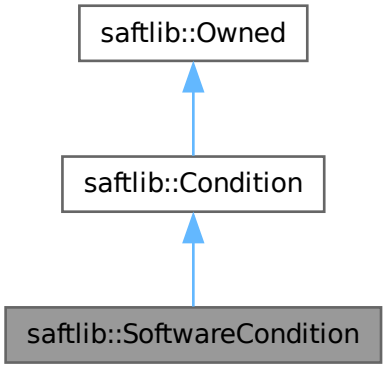
- src/SoftwareActionSink_Service.hpp
- src/SoftwareActionSink_Service.cpp

7.136 saftlib::SoftwareCondition Class Reference

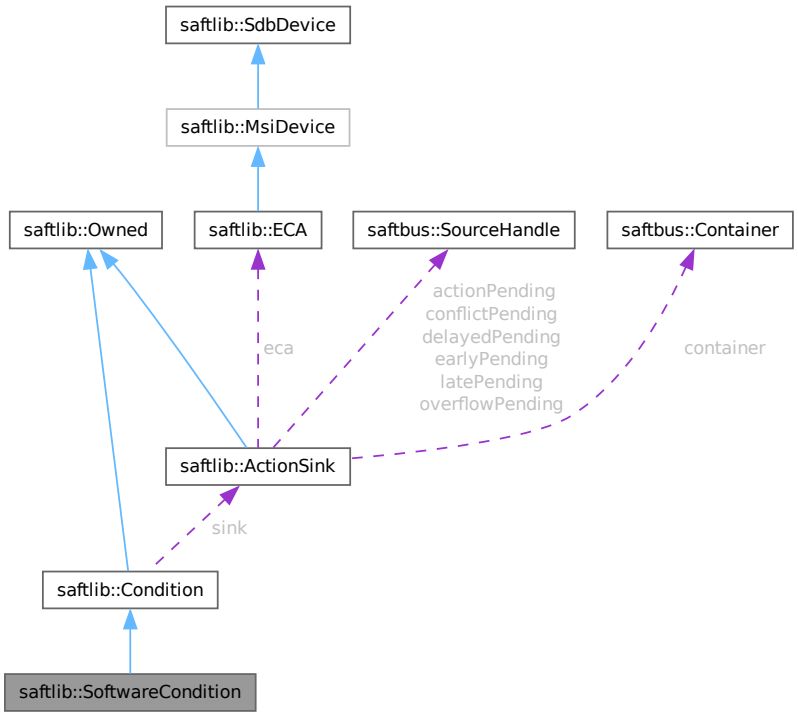
Matched against incoming events on a SoftwareActionSink.

`#include <SoftwareCondition.hpp>`

Inheritance diagram for saftlib::SoftwareCondition:



Collaboration diagram for saftlib::SoftwareCondition:



Public Types

- typedef SoftwareCondition_Service **ServiceType**

Public Member Functions

- **SoftwareCondition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, saftbus::Container *container)

Public Member Functions inherited from saftlib::Condition

- **Condition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)
- uint64_t getID () const

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask () const

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset () const

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate () const

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly () const

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict () const

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed () const

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive () const

  The condition should be actively matched against events.
- void **setActive** (bool val)
- std::string & **getObjectPath** ()
- uint32_t **getRawTag** () const
- void **setRawActive** (bool val)
- unsigned **getNumber** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Attributes

- sigc::signal< void, uint64_t, uint64_t, saftlib::Time, saftlib::Time, uint16_t > SigAction

  Emitted whenever the condition matches a timing event.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Additional Inherited Members

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes inherited from saftlib::Condition

- std::string **objectPath**
- ActionSink * **sink**
- unsigned **number**
- uint64_t **id**
- uint64_t **mask**
- int64_t **offset**
- uint32_t **tag**
- bool **acceptLate**
- bool **acceptEarly**
- bool **acceptConflict**
- bool **acceptDelayed**
- bool **active**

7.136.1 Detailed Description

Matched against incoming events on a SoftwareActionSink.

de.gsi.saftlib.SoftwareCondition

SoftwareConditions are created by SoftwareActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

7.136.2 Member Data Documentation

7.136.2.1 SigAction

```
sigc::signal<void, uint64_t, uint64_t, saftlib::Time, saftlib::Time, uint16_t > saftlib::↩
SoftwareCondition::SigAction
```

Emitted whenever the condition matches a timing event.

Parameters

event	The event identifier that matched this rule.
param	The parameter field, whose meaning depends on the event ID.
deadline	The scheduled execution timestamp of the action (event time + offset).
executed	The actual execution timestamp of the action.
flags	Whether the action was (ok=0,late=1,early=2,conflict=4,delayed=8)

While the underlying hardware strives to deliver the action precisely on the deadline, the software stack adds non-deterministic delay, so the deadline may be milliseconds in the past. The late flag only indicates that the hardware failed to meet the required deadline. Similarly, the executed timestamp is the time when the hardware delivered the action, not when software has received it. Actions with error flags (late, early, conflict, delayed) are only delivered to this signal if the Condition which generated them specified that the respective error should be accepted.

The documentation for this class was generated from the following files:

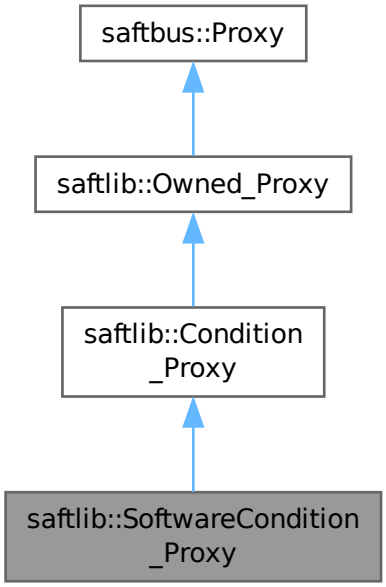
- src/SoftwareCondition.hpp
- src/SoftwareCondition.cpp

7.137 saftlib::SoftwareCondition_Proxy Class Reference

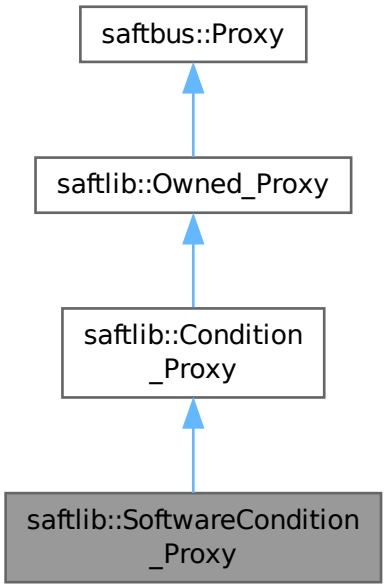
Matched against incoming events on a SoftwareActionSink.

```
#include <SoftwareCondition_Proxy.hpp>
```

Inheritance diagram for saftlib::SoftwareCondition_Proxy:



Collaboration diagram for saftlib::SoftwareCondition_Proxy:



Public Member Functions

- **SoftwareCondition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::↩ SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Public Member Functions inherited from saftlib::Condition_Proxy

- **Condition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t getID ()

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask ()

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset ()

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate ()

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly ()

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict ()

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed ()

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive ()

  The condition should be actively matched against events.
- void **setActive** (bool val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.

- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< SoftwareCondition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Condition_Proxy

- static std::shared_ptr< Condition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Public Attributes

- sigc::signal< void, uint64_t, uint64_t, saftlib::Time, saftlib::Time, uint16_t > SigAction

  Emitted whenever the condition matches a timing event.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.137.1 Detailed Description

Matched against incoming events on a SoftwareActionSink.

de.gsi.saftlib.SoftwareCondition

SoftwareConditions are created by SoftwareActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

7.137.2 Member Function Documentation

7.137.2.1 signal_dispatch()

```
bool saftlib::SoftwareCondition_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Reimplemented from saftlib::Condition_Proxy.

7.137.3 Member Data Documentation

7.137.3.1 SigAction

```
sigc::signal<void, uint64_t, uint64_t, saftlib::Time, saftlib::Time, uint16_t> saftlib::↩
SoftwareCondition_Proxy::SigAction
```

Emitted whenever the condition matches a timing event.

Parameters

event	The event identifier that matched this rule.
param	The parameter field, whose meaning depends on the event ID.
deadline	The scheduled execution timestamp of the action (event time + offset).
executed	The actual execution timestamp of the action.
flags	Whether the action was (ok=0,late=1,early=2,conflict=4,delayed=8)

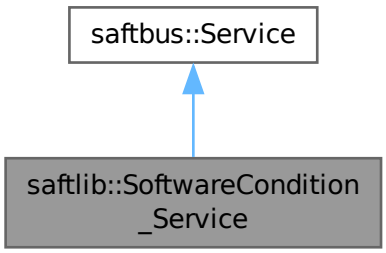
While the underlying hardware strives to deliver the action precisely on the deadline, the software stack adds non-deterministic delay, so the deadline may be milliseconds in the past. The late flag only indicates that the hardware failed to meet the required deadline. Similarly, the executed timestamp is the time when the hardware delivered the action, not when software has received it. Actions with error flags (late, early, conflict, delayed) are only delivered to this signal if the Condition which generated them specified that the respective error should be accepted.

The documentation for this class was generated from the following files:

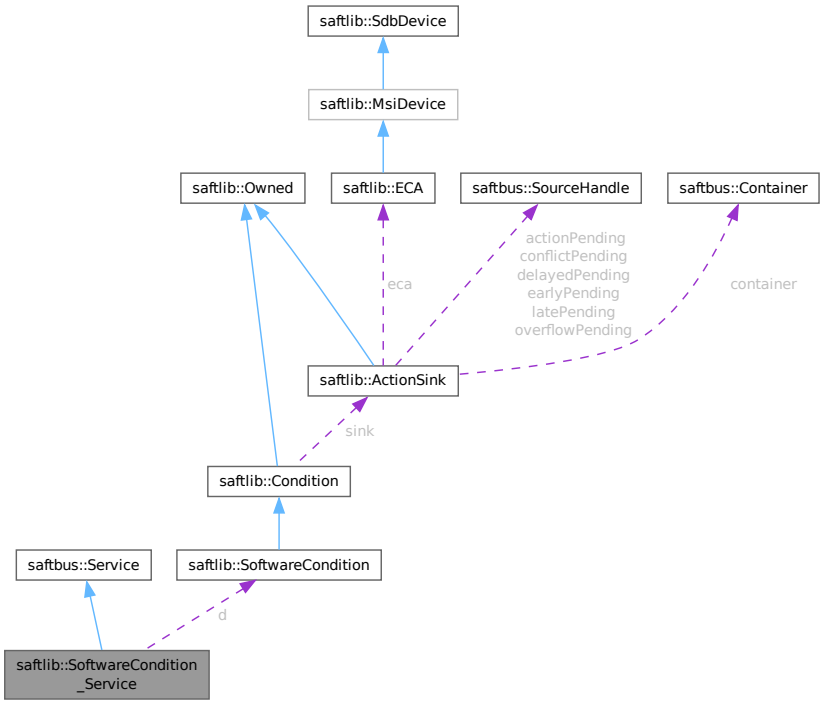
- src/SoftwareCondition_Proxy.hpp
- src/SoftwareCondition_Proxy.cpp

7.138 saftlib::SoftwareCondition_Service Class Reference

Inheritance diagram for saftlib::SoftwareCondition_Service:



Collaboration diagram for saftlib::SoftwareCondition_Service:



Public Types

- typedef SoftwareCondition **DriverType**

Public Member Functions

- **SoftwareCondition_Service** (SoftwareCondition *instance, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **SigAction_dispatch_function** (uint64_t arg_1, uint64_t arg_2, saftlib::Time arg_3, saftlib::Time arg_4, uint16_t arg_5)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

execute one of the functions in one of the interfaces of the derived class.

- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- SoftwareCondition * **d**
- sigc::connection **SigAction_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.138.1 Member Function Documentation

7.138.1.1 call()

```
void saftlib::SoftwareCondition_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/SoftwareCondition_Service.hpp
- src/SoftwareCondition_Service.cpp

7.139 software_tr::SoftwareECA Struct Reference

Classes

- struct Event
- struct Search
- struct SearchCandidate
- struct Walker

Public Member Functions

- void **inject** ()
- bool **is_prefix_mask** (uint64_t mask)
- uint64_t **fix_mask** (uint64_t mask)
- void **push_search** (const Search &s)
- void **analyse_search_candidates_normal** ()
- void **analyse_search_candidates_zero** ()
- void **analyse_search_candidates** ()

Static Public Member Functions

- static uint64_t **get_time_ns** ()
- static void **eca_events_to_actions** (SoftwareECA *software_eca)

Public Attributes

- std::vector< SearchCandidate > **search_candidates**
- std::vector< Search > **searches**
- std::map< int, Walker > **walker**
- uint32_t **in_buffer** [8]
- uint32_t **msi_target_adr**
- std::mutex **events_mutex**
- std::mutex **actions_mutex**
- std::deque< Event > **events**
- std::deque< Event > **actions**

The documentation for this struct was generated from the following file:

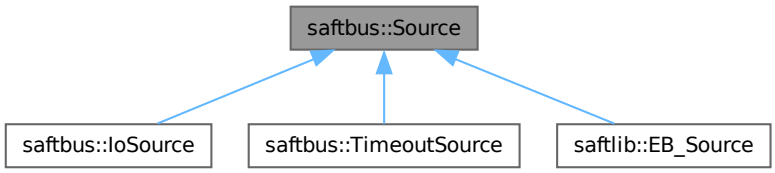
- src/saft-software-tr.cpp

7.140 saftbus::Source Class Reference

Base class of all event sources in a saftbus::Loop.

`#include <loop.hpp>`

Inheritance diagram for saftbus::Source:



Public Member Functions

- virtual bool **prepare** (std::chrono::milliseconds &timeout_ms)=0
- virtual bool **check** ()=0
- virtual bool **dispatch** ()=0
- virtual std::string **type** ()=0
- long **get_id** ()

Protected Member Functions

- void **add_poll** (pollfd *pfd)
- void **remove_poll** (pollfd *pfd)
- void **clear_poll** ()

Friends

- class **Loop**

7.140.1 Detailed Description

Base class of all event sources in a saftbus::Loop.

The documentation for this class was generated from the following files:

- saftbus/loop.hpp
- saftbus/loop.cpp

7.141 saftbus::SourceHandle Class Reference

unique identifier for an event source in a saftbus::Loop

`#include <loop.hpp>`

Public Member Functions

- long **get_source_id** () const
- long **get_loop_id** () const
- bool **connected** () const

Friends

- class **Loop**

7.141.1 Detailed Description

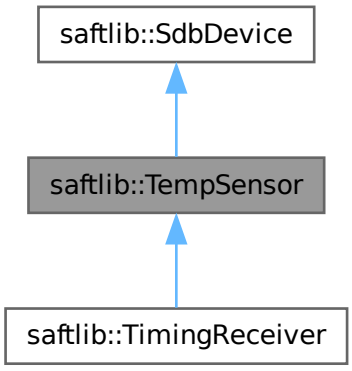
unique identifier for an event source in a saftbus::Loop

The documentation for this class was generated from the following file:

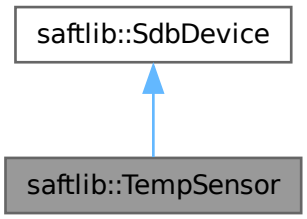
- saftbus/loop.hpp

7.142 saftlib::TempSensor Class Reference

Inheritance diagram for saftlib::TempSensor:



Collaboration diagram for saftlib::TempSensor:



Public Member Functions

- **TempSensor** (etherbone::Device &device)
- bool getTemperatureSensorAvail () const

  Check if a temperature sensor is available.
- int32_t CurrentTemperature ()

  The current temperature in degree Celsius.

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.142.1 Member Function Documentation

7.142.1.1 CurrentTemperature()

```
int32_t saftlib::TempSensor::CurrentTemperature ( )
```

The current temperature in degree Celsius.

Returns

Temperature in degree Celsius.

The valid temperature range is from -70 to 127 degree Celsius. The data type is 32-bit signed integer.

7.142.1.2 getTemperatureSensorAvail()

```
bool saftlib::TempSensor::getTemperatureSensorAvail ( ) const
```

Check if a temperature sensor is available.

Returns

Check if a temperature sensor is available

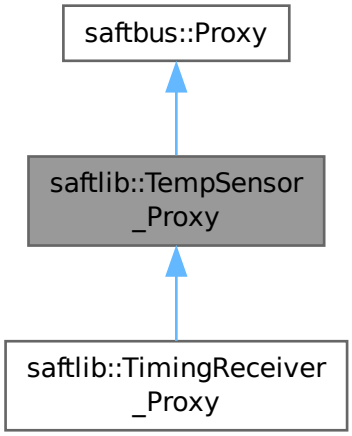
in a timing receiver.

The documentation for this class was generated from the following files:

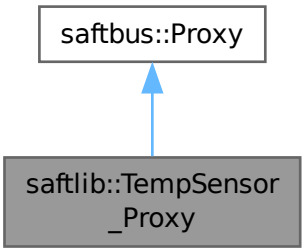
- src/TempSensor.hpp
- src/TempSensor.cpp

7.143 saftlib::TempSensor_Proxy Class Reference

Inheritance diagram for saftlib::TempSensor_Proxy:



Collaboration diagram for saftlib::TempSensor_Proxy:



Public Member Functions

- **TempSensor_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- bool getTemperatureSensorAvail ()

  Check if a temperature sensor is available.
- int32_t CurrentTemperature ()

  The current temperature in degree Celsius.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< TempSensor_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.143.1 Member Function Documentation

7.143.1.1 CurrentTemperature()

```
int32_t saftlib::TempSensor_Proxy::CurrentTemperature ( )
```

The current temperature in degree Celsius.

Returns

Temperature in degree Celsius.

The valid temperature range is from -70 to 127 degree Celsius. The data type is 32-bit signed integer.

7.143.1.2 getTemperatureSensorAvail()

```
bool saftlib::TempSensor_Proxy::getTemperatureSensorAvail ( )
```

Check if a temperature sensor is available.

Returns

Check if a temperature sensor is available

in a timing receiver.

7.143.1.3 signal_dispatch()

```
bool saftlib::TempSensor_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

Implements saftbus::Proxy.

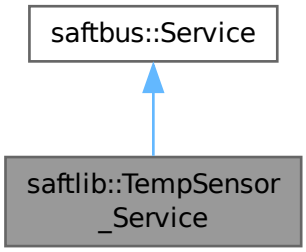
Reimplemented in saftlib::TimingReceiver_Proxy.

The documentation for this class was generated from the following files:

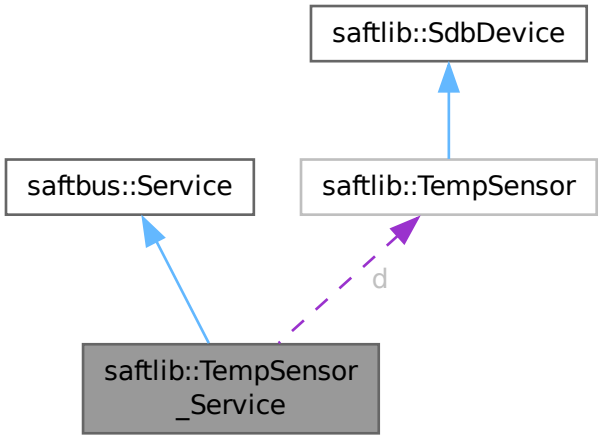
- src/TempSensor_Proxy.hpp
- src/TempSensor_Proxy.cpp

7.144 saftlib::TempSensor_Service Class Reference

Inheritance diagram for saftlib::TempSensor_Service:



Collaboration diagram for saftlib::TempSensor_Service:



Public Types

- typedef TempSensor **DriverType**

Public Member Functions

- **TempSensor_Service** (TempSensor *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- TempSensor * **d**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.144.1 Member Function Documentation

7.144.1.1 call()

```
void saftlib::TempSensor_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/TempSensor_Service.hpp
- src/TempSensor_Service.cpp

7.145 saftlib::Time Class Reference

Public Member Functions

- **Time** (const Time &t)
- uint64_t **getUTC** () const
- uint64_t **getTAI** () const
- Time & **operator=** (const Time &t)
- Time & **operator+=** (const int64_t &duration)
- Time & **operator-=** (const int64_t &duration)
- int64_t **operator-** (const Time &rhs) const
- int64_t **getUTCOffset** () const
- bool **operator>** (const Time &rhs) const
- bool **operator<** (const Time &rhs) const
- bool **operator>=** (const Time &rhs) const
- bool **operator<=** (const Time &rhs) const
- bool **operator==** (const Time &rhs) const
- bool **operator!=** (const Time &rhs) const
- int **isLeapUTC** () const

Friends

- Time **makeTimeUTC** (uint64_t UTC, bool isLeap)
- Time **makeTimeTAI** (uint64_t TAI)

The documentation for this class was generated from the following file:

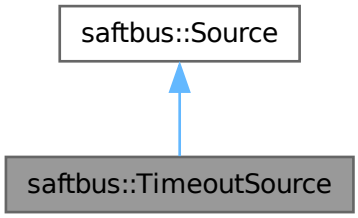
- src/Time.hpp

7.146 saftbus::TimeoutSource Class Reference

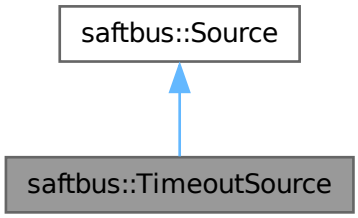
An event source that is active after a given amount of time has passed.

`#include <loop.hpp>`

Inheritance diagram for saftbus::TimeoutSource:



Collaboration diagram for saftbus::TimeoutSource:



Public Member Functions

- TimeoutSource (std::function< bool(void)> slot, std::chrono::milliseconds interval, std::chrono::milliseconds offset)
- TimeoutSource (std::function< bool(void)> slot, std::chrono::milliseconds interval)
- bool prepare (std::chrono::milliseconds &timeout_ms) override
- bool check () override
- bool dispatch () override
- std::string type () override

Public Member Functions inherited from saftbus::Source

- virtual bool **prepare** (std::chrono::milliseconds &timeout_ms)=0
- virtual bool **check** ()=0
- virtual bool **dispatch** ()=0
- virtual std::string **type** ()=0
- long **get_id** ()

Additional Inherited Members

Protected Member Functions inherited from saftbus::Source

- void **add_poll** (pollfd *pfd)
- void **remove_poll** (pollfd *pfd)
- void **clear_poll** ()

7.146.1 Detailed Description

An event source that is active after a given amount of time has passed.

The source is removed whenever the connected function returns false.

7.146.2 Constructor & Destructor Documentation

7.146.2.1 TimeoutSource() [1/2]

```
saftbus::TimeoutSource::TimeoutSource (
            std::function< bool(void)> slot,
            std::chrono::milliseconds interval,
            std::chrono::milliseconds offset )
```

Parameters

slot	the function that is called periodically.
interval	duration between calls to slot. If interval is smaller than 1 ms, it is set to 1 ms.
offset	fist execution starts after waiting for offset amount of time.

7.146.2.2 TimeoutSource() [2/2]

```
saftbus::TimeoutSource::TimeoutSource (
            std::function< bool(void)> slot,
            std::chrono::milliseconds interval )
```

Parameters

slot	the function that is called periodically. If interval is smaller than 1 ms, it is set to 1 ms.
interval	duration between calls to slot, fist execution starts at after waiting one interval worth of time.

7.146.3 Member Function Documentation

7.146.3.1 check()

```
bool saftbus::TimeoutSource::check ( )  [override], [virtual]
```

Implements saftbus::Source.

7.146.3.2 dispatch()

```
bool saftbus::TimeoutSource::dispatch ( )  [override], [virtual]
```

Implements saftbus::Source.

7.146.3.3 prepare()

```
bool saftbus::TimeoutSource::prepare (
            std::chrono::milliseconds & timeout_ms )  [override], [virtual]
```

Implements saftbus::Source.

7.146.3.4 type()

```
std::string saftbus::TimeoutSource::type ( )  [override], [virtual]
```

Implements saftbus::Source.

The documentation for this class was generated from the following files:

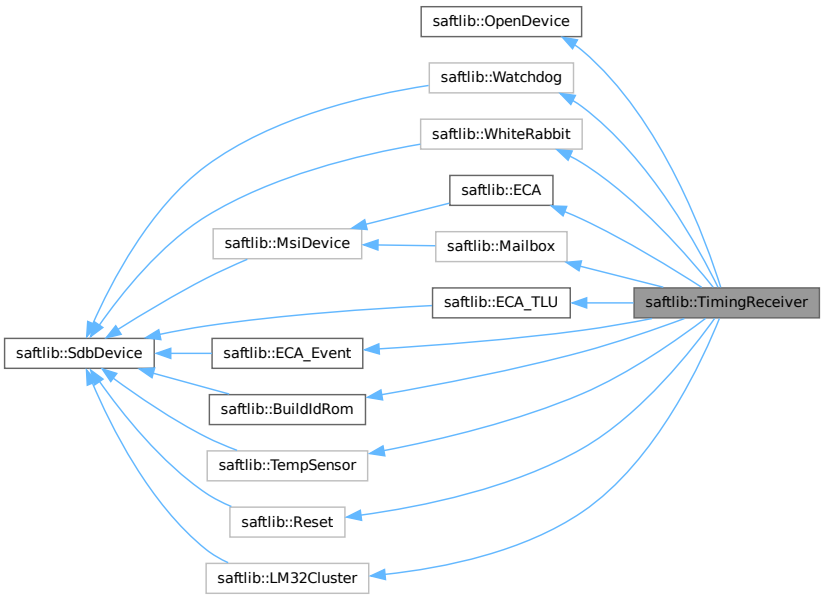
- saftbus/loop.hpp
- saftbus/loop.cpp

7.147 saftlib::TimingReceiver Class Reference

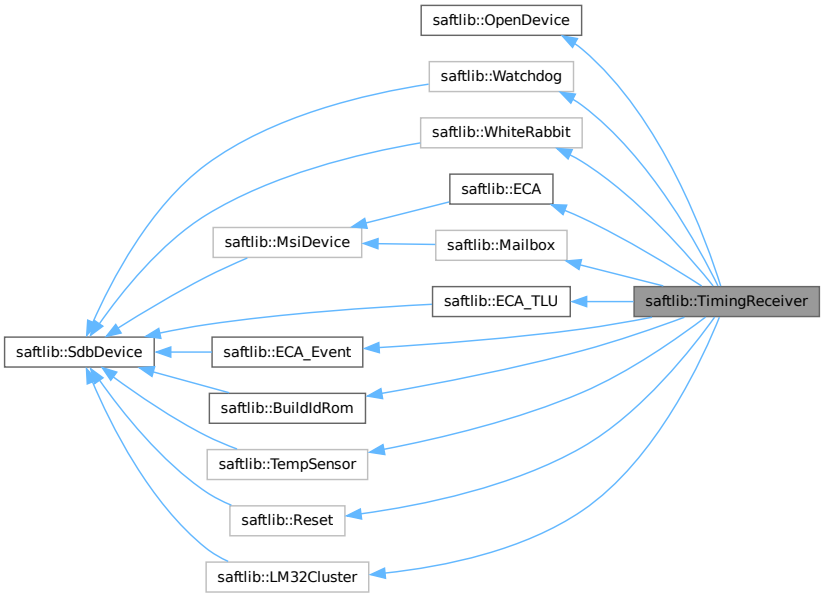
A timing receiver.

`#include <TimingReceiver.hpp>`

Inheritance diagram for saftlib::TimingReceiver:



Collaboration diagram for saftlib::TimingReceiver:



Public Member Functions

- **TimingReceiver** (SAFTd &saftd, const std::string &name, const std::string &etherbone_path, int polling_↩ interval_ms=1, saftbus::Container *container=nullptr)
- const std::string & **getObjectPath** () const
- void **Remove** ()

  Remove the device from saftlib management.
- std::string getName () const

  The logical name with which the device was connected.
- saftlib::Time CurrentTime () const

  The current time of the timingreceiver.
- void InjectEvent (uint64_t event, uint64_t param, saftlib::Time time) const

  Simulate the receipt of a timing event.
- std::map< std::string, std::map< std::string, std::string > > getInterfaces () const

  List of all object instances of various hardware.
- void installAddon (const std::string &interface_name, std::unique_ptr< TimingReceiverAddon > addon)

  saftbus plugins may extend the functionality of a TimingReceiver. They should use this method to publish the additional interfaces.
- void **removeAddon** (const std::string &interface_name)

Public Member Functions inherited from saftlib::OpenDevice

- OpenDevice (const etherbone::Socket &socket, const std::string &etherbone_path, int polling_interval_ms=1, SAFTd *saftd=nullptr)

  open given etherbone_path on given socket.
- etherbone::Device & **get_device** ()
- std::string getEtherbonePath () const

  The path through which the device is reached.
- std::string getEbForwardPath () const

  If the device is not capable of multiplexing multiple users a /dev/pts/<num> device is created that can be used with eb-tools.

Public Member Functions inherited from saftlib::Watchdog

- **Watchdog** (etherbone::Device &device)
- bool **aquire** ()
- void **update** ()

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Public Member Functions inherited from saftlib::WhiteRabbit

- **WhiteRabbit** (etherbone::Device &device)
- bool getLocked () const

  The timing receiver is locked to the timing grandmaster.

Public Member Functions inherited from saftlib::ECA

- const std::string & **get_object_path** ()
- etherbone::Device & **get_device** ()
- void **compile** ()
- **ECA** (SAFTd &saftd, etherbone::Device &device, const std::string &object_path, saftbus::Container ∗container)
- bool **addActionSink** (int channel, std::unique_ptr< ActionSink > sink)

  add sink and let ECA take ownership of the sink object
- uint16_t **updateMostFull** (unsigned channel)
- void **removeSowftwareActionSink** (SoftwareActionSink ∗sas)
- uint64_t ReadRawCurrentTime () const

  The current time of the timingreceiver.
- std::string NewSoftwareActionSink (const std::string &name)

  Create a new SoftwareActionSink.
- std::map< std::string, std::string > getSoftwareActionSinks () const

  A list of all current SoftwareActionSinks.
- std::map< std::string, std::string > getSCUbusActionSinks () const

  A list of all SCUbusActionSinks.
- std::map< std::string, std::string > getEmbeddedCPUActionSinks () const

  A list of all EmbeddedCPUActionSinks.
- std::map< std::string, std::string > getWbmActionSinks () const

  A list of all WbmActionSinks.
- SoftwareActionSink ∗ getSoftwareActionSink (const std::string &sas_obj_path)

  get a pointer to a SoftwareActionSink in a stand-alone application
- void **ToggleActive** ()

  activate all inactive owned conditions and inactivate all active owned conditions
- void **InactivateAll** ()

  deactivate all owned conditions
- std::map< std::string, std::string > getOutputs () const

  A list of all the high/low outputs on the receiver.
- uint32_t getFree () const

  The number of additional conditions that may be activated.
- void **resetMostFull** (unsigned channel)
- Output ∗ **getOutput** (const std::string &output_obj_path)

Public Member Functions inherited from saftlib::MsiDevice

- **MsiDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID)

Public Member Functions inherited from saftlib::ECA_TLU

- **ECA_TLU** (etherbone::Device &device, saftbus::Container *container=nullptr)
- void **addInput** (std::unique_ptr< Input > input)

  add source and let ECA take ownership of the sink object
- std::map< std::string, std::string > **getInputs** () const
- void **configInput** (unsigned channel, bool enable, uint64_t event, uint32_t stable)

Public Member Functions inherited from saftlib::ECA_Event

- **ECA_Event** (etherbone::Device &device, saftbus::Container *container=nullptr)
- void InjectEventRaw (uint64_t event, uint64_t param, uint64_t time) const

  Simulate the receipt of a timing event.

Public Member Functions inherited from saftlib::BuildIdRom

- **BuildIdRom** (etherbone::Device &device)
- std::map< std::string, std::string > getGatewareInfo () const

  Key-value map of hardware build information.
- std::string getGatewareVersion () const

  Hardware build version.

Public Member Functions inherited from saftlib::TempSensor

- **TempSensor** (etherbone::Device &device)
- bool getTemperatureSensorAvail () const

  Check if a temperature sensor is available.
- int32_t CurrentTemperature ()

  The current temperature in degree Celsius.

Public Member Functions inherited from saftlib::Reset

- **Reset** (etherbone::Device &device)
- void WdRetrigger ()

  retrigger reset watchdog.
- void CpuHalt (unsigned idx)

  permanently assert reset line of cpu[idx]
- void CpuReset (unsigned idx)

  release reset line of cpu[idx]
- uint32_t CpuHaltStatus ()

  get the 'halt status' of all user lm32 (rightmost bit: CPU 0). bit='1' means halted.

Public Member Functions inherited from saftlib::Mailbox

- void UseSlot (int slot_index, uint32_t value)

  write a value to the preconfigured address.
- **Mailbox** (etherbone::Device &device)
- std::unique_ptr< Slot > ConfigureSlot (uint32_t target_address)

  find a free slot in the mailbox and configure it with target_address

Public Member Functions inherited from saftlib::LM32Cluster

- **LM32Cluster** (etherbone::Device &device, TimingReceiver *tr)
- unsigned getCpuCount ()

  number of CPUs
- void SafeHaltCpu (unsigned cpu_idx)

  stop execution of cpu[cpu_idx]
- void **WriteFirmware** (unsigned cpu_idx, const std::string &filename)

Additional Inherited Members

Public Attributes inherited from saftlib::WhiteRabbit

- sigc::signal< void, bool > **Locked**

Public Attributes inherited from saftlib::LM32Cluster

- std::vector< eb_address_t > **dpram_lm32_adr_first**
- std::vector< eb_address_t > **dpram_lm32_adr_last**

Protected Attributes inherited from saftlib::OpenDevice

- std::string **etherbone_path**
- struct stat **dev_stat**
- etherbone::Device **device**

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

Protected Attributes inherited from saftlib::WhiteRabbit

- bool **locked**

Protected Attributes inherited from saftlib::MsiDevice

- etherbone::sdb_msi_device **msi_device**

7.147.1 Detailed Description

A timing receiver.

de.gsi.saftlib.TimingReceiver:

Parameters

saftd	A pointer to a SAFTd object. The SAFTd object holds the etherbone socket on which the MSI arrive.
name	The logical name of the Timing receiver
etherbone_path	The etherbone path of the hardware
container	A pointer to a saftbus container. If an instance of TimingReceiver should run on a saftbus deamon, container must point to the saftbus::Container instance. If the TimingReceiver is used stand-alone, this pointer must be nullptr.

Timing receivers are attached to saftlib by specifying a name and an etherbone path. The name should denote the logical relationship of the device to saftd. For example, baseboard would be a good name for the timing receiver attached to an SCU. If an exploder is being used to output events to an oscilloscope, a good logical name might be scope. In these examples, the path for the SCU baseboard would be dev/wbm0, and the scope exploder would be dev/ttyUSB3 or similar. This scheme is intended to make it easy to hot-swap hardware. If the exploder dies, you can simply attach a new one under the same logical name, even though the path might be different.

Timing receivers can respond to timing events from the data master. The can also respond to external timing triggers via inputs. The general idea is that a TimingReceiver has ActionSinks to which it sends actions in response to incoming timing events. Timing events are matched with Conditions to create the Actions sent to the Sinks. EventSources are objects which create timing events, to be matched by the conditions. The data master is a global EventSource to which all TimingReceivers listen. However, external inputs can also be configured to generate timing events. Furthermore, a TimingReceiver can simulate the receipt of a timing event by calling the InjectEvent method.

Timing receivers always typically have binary outputs lines (OutputActionSinks), which are listed in the Outputs property. Similarly, they often have digital inputs (InputEventSources). Some timing receivers have special purpose interfaces. For example, an SCU has the SCUbusActionSink which generates 32-bit messages over the SCU backplane. These special interfaces can be found in the interfaces property. The SCU backplane would be found under the SCUbusActionSink key, and as there is only one, it would be the 0th.

7.147.2 Member Function Documentation

7.147.2.1 CurrentTime()

```
saftlib::Time saftlib::TimingReceiver::CurrentTime ( ) const
```

The current time of the timingreceiver.

Returns

the current time of the timingreceiver

The result type is saftlib::Time, which can be used to obtain either the number of nanoseconds since 1970, or the same value minus the current UTC offset. Due to delays in software, the returned value is probably several milliseconds behind the true time.

7.147.2.2 getInterfaces()

```
std::map< std::string, std::map< std::string, std::string > > saftlib::TimingReceiver::get↩
Interfaces ( ) const
```

List of all object instances of various hardware.

Returns

List of all object instances of various hardware.

The key in the dictionary is the name of the interface. The value is all object paths to hardware implementing that interface.

7.147.2.3 getName()

```
std::string saftlib::TimingReceiver::getName ( ) const
```

The logical name with which the device was connected.

Returns

The logical name with which the device was connected.

7.147.2.4 InjectEvent()

```
void saftlib::TimingReceiver::InjectEvent (
            uint64_t event,
            uint64_t param,
            saftlib::Time time ) const
```

Simulate the receipt of a timing event.

Parameters

event	The event identifier which is matched against Conditions
param	The parameter field, whose meaning depends on the event ID.
time	The execution time for the event, added to condition offsets.

Sometimes it is useful to simulate the receipt of a timing event. This allows software to test that configured conditions lead to the desired behaviour without needing the data master to send anything.

7.147.2.5 installAddon()

```
void saftlib::TimingReceiver::installAddon (
            const std::string & interface_name,
            std::unique_ptr< TimingReceiverAddon > addon )
```

saftbus plugins may extend the functionality of a TimingReceiver. They should use this method to publish the additional interfaces.

Parameters

interface_name	the name of the addon
name	and object path of instances of interface

The documentation for this class was generated from the following files:

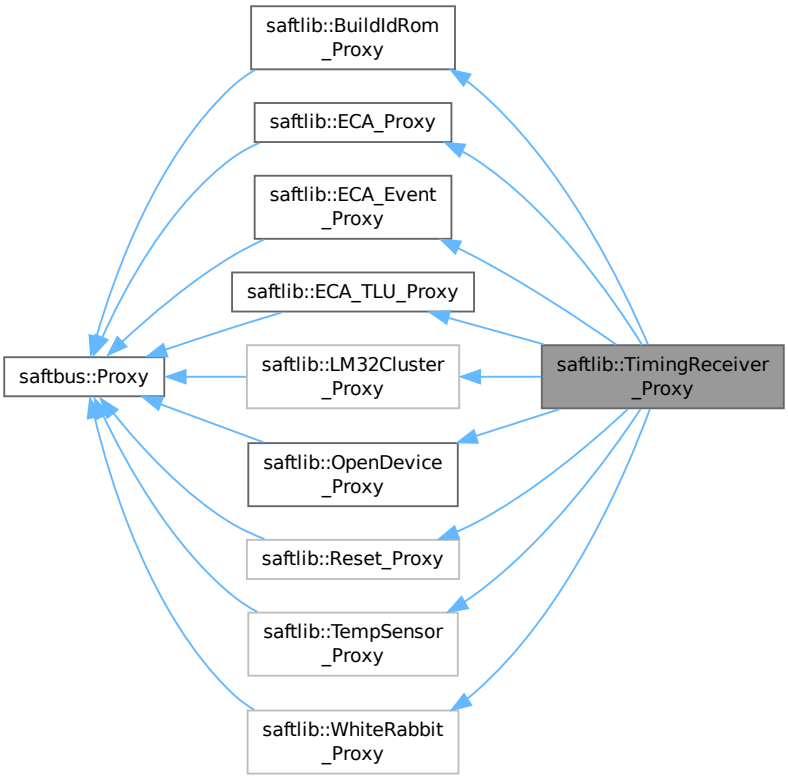
- src/TimingReceiver.hpp
- src/TimingReceiver.cpp

7.148 saftlib::TimingReceiver_Proxy Class Reference

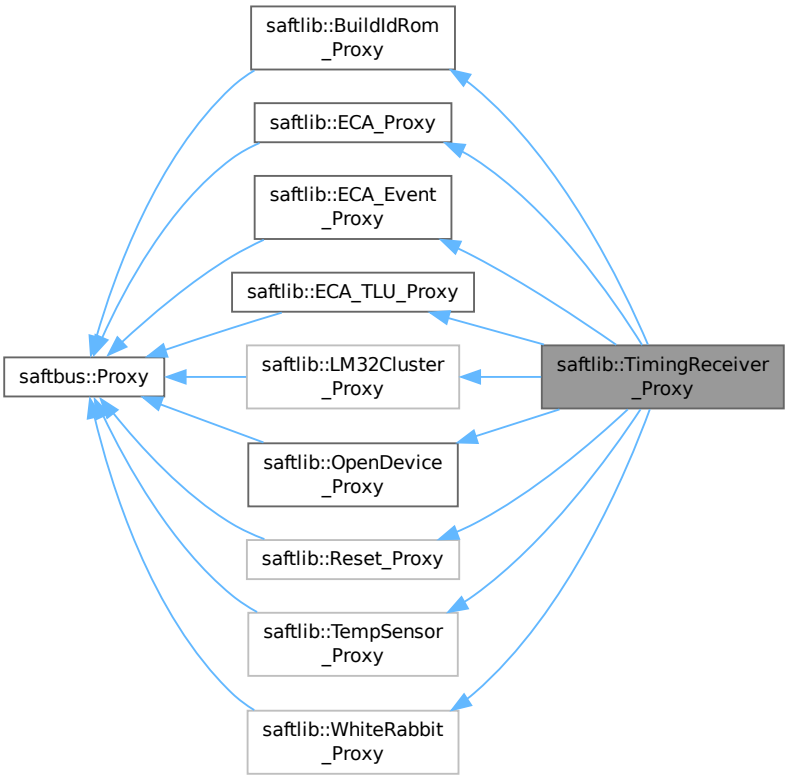
A timing receiver.

`#include <TimingReceiver_Proxy.hpp>`

Inheritance diagram for saftlib::TimingReceiver_Proxy:



Collaboration diagram for saftlib::TimingReceiver_Proxy:



Public Member Functions

- **TimingReceiver_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::↩ SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void **Remove** ()

  Remove the device from saftlib management.
- std::string getName ()

  The logical name with which the device was connected.
- saftlib::Time CurrentTime ()

  The current time of the timingreceiver.
- void InjectEvent (uint64_t event, uint64_t param, saftlib::Time time)

  Simulate the receipt of a timing event.
- std::map< std::string, std::map< std::string, std::string > > getInterfaces ()

  List of all object instances of various hardware.

Public Member Functions inherited from saftlib::BuildIdRom_Proxy

- **BuildIdRom_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())

- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::map< std::string, std::string > getGatewareInfo ()

  Key-value map of hardware build information.
- std::string getGatewareVersion ()

  Hardware build version.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Public Member Functions inherited from saftlib::ECA_Proxy

- **ECA_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t ReadRawCurrentTime ()

  The current time of the timingreceiver.
- std::string NewSoftwareActionSink (const std::string &name)

  Create a new SoftwareActionSink.
- std::map< std::string, std::string > getSoftwareActionSinks ()

  A list of all current SoftwareActionSinks.
- std::map< std::string, std::string > getSCUbusActionSinks ()

  A list of all SCUbusActionSinks.
- std::map< std::string, std::string > getEmbeddedCPUActionSinks ()

  A list of all EmbeddedCPUActionSinks.
- std::map< std::string, std::string > getWbmActionSinks ()

  A list of all WbmActionSinks.
- void **ToggleActive** ()

  activate all inactive owned conditions and inactivate all active owned conditions
- void **InactivateAll** ()

  deactivate all owned conditions
- std::map< std::string, std::string > getOutputs ()

  A list of all the high/low outputs on the receiver.
- uint32_t getFree ()

  The number of additional conditions that may be activated.

Public Member Functions inherited from saftlib::ECA_Event_Proxy

- **ECA_Event_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void InjectEventRaw (uint64_t event, uint64_t param, uint64_t time)

  Simulate the receipt of a timing event.

Public Member Functions inherited from saftlib::ECA_TLU_Proxy

- **ECA_TLU_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::map< std::string, std::string > **getInputs** ()

Public Member Functions inherited from saftlib::LM32Cluster_Proxy

- **LM32Cluster_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- unsigned getCpuCount ()

  number of CPUs
- void SafeHaltCpu (unsigned cpu_idx)

  stop execution of cpu[cpu_idx]

Public Member Functions inherited from saftlib::OpenDevice_Proxy

- **OpenDevice_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::string getEtherbonePath ()

  The path through which the device is reached.
- std::string getEbForwardPath ()

  If the device is not capable of multiplexing multiple users a /dev/pts/<num> device is created that can be used with eb-tools.

Public Member Functions inherited from saftlib::Reset_Proxy

- **Reset_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void WdRetrigger ()

  retrigger reset watchdog.
- void CpuHalt (unsigned idx)

  permanently assert reset line of cpu[idx]
- void CpuReset (unsigned idx)

  release reset line of cpu[idx]
- uint32_t CpuHaltStatus ()

  get the 'halt status' of all user lm32 (rightmost bit: CPU 0). bit='1' means halted.

Public Member Functions inherited from saftlib::TempSensor_Proxy

- **TempSensor_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- bool getTemperatureSensorAvail ()

  Check if a temperature sensor is available.
- int32_t CurrentTemperature ()

  The current temperature in degree Celsius.

Public Member Functions inherited from saftlib::WhiteRabbit_Proxy

- **WhiteRabbit_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- bool getLocked ()

  The timing receiver is locked to the timing grandmaster.

Static Public Member Functions

- static std::shared_ptr< TimingReceiver_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::BuildIdRom_Proxy

- static std::shared_ptr< BuildIdRom_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ECA_Proxy

- static std::shared_ptr< ECA_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ECA_Event_Proxy

- static std::shared_ptr< ECA_Event_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ECA_TLU_Proxy

- static std::shared_ptr< ECA_TLU_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::LM32Cluster_Proxy

- static std::shared_ptr< LM32Cluster_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::OpenDevice_Proxy

- static std::shared_ptr< OpenDevice_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Reset_Proxy

- static std::shared_ptr< Reset_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::TempSensor_Proxy

- static std::shared_ptr< TempSensor_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::WhiteRabbit_Proxy

- static std::shared_ptr< WhiteRabbit_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::WhiteRabbit_Proxy

- sigc::signal< void, bool > **Locked**

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.148.1 Detailed Description

A timing receiver.

de.gsi.saftlib.TimingReceiver:

Parameters

saftd	A pointer to a SAFTd object. The SAFTd object holds the etherbone socket on which the MSI arrive.
name	The logical name of the Timing receiver
etherbone_path	The etherbone path of the hardware
container	A pointer to a saftbus container. If an instance of TimingReceiver should run on a saftbus deamon, container must point to the saftbus::Container instance. If the TimingReceiver is used stand-alone, this pointer must be nullptr.

Timing receivers are attached to saftlib by specifying a name and an etherbone path. The name should denote the logical relationship of the device to saftd. For example, baseboard would be a good name for the timing receiver attached to an SCU. If an exploder is being used to output events to an oscilloscope, a good logical name might be scope. In these examples, the path for the SCU baseboard would be dev/wbm0, and the scope exploder would be dev/ttyUSB3 or similar. This scheme is intended to make it easy to hot-swap hardware. If the exploder dies, you can simply attach a new one under the same logical name, even though the path might be different.

Timing receivers can respond to timing events from the data master. The can also respond to external timing triggers via inputs. The general idea is that a TimingReceiver has ActionSinks to which it sends actions in response to incoming timing events. Timing events are matched with Conditions to create the Actions sent to the Sinks. EventSources are objects which create timing events, to be matched by the conditions. The data master is a global EventSource to which all TimingReceivers listen. However, external inputs can also be configured to generate timing events. Furthermore, a TimingReceiver can simulate the receipt of a timing event by calling the InjectEvent method.

Timing receivers always typically have binary outputs lines (OutputActionSinks), which are listed in the Outputs property. Similarly, they often have digital inputs (InputEventSources). Some timing receivers have special purpose interfaces. For example, an SCU has the SCUbusActionSink which generates 32-bit messages over the SCU backplane. These special interfaces can be found in the interfaces property. The SCU backplane would be found under the SCUbusActionSink key, and as there is only one, it would be the 0th.

7.148.2 Member Function Documentation

7.148.2.1 CurrentTime()

```
saftlib::Time saftlib::TimingReceiver_Proxy::CurrentTime ( )
```

The current time of the timingreceiver.

Returns

the current time of the timingreceiver

The result type is saftlib::Time, which can be used to obtain either the number of nanoseconds since 1970, or the same value minus the current UTC offset. Due to delays in software, the returned value is probably several milliseconds behind the true time.

7.148.2.2 getInterfaces()

```
std::map< std::string, std::map< std::string, std::string > > saftlib::TimingReceiver_Proxy↩
::getInterfaces ( )
```

List of all object instances of various hardware.

Returns

List of all object instances of various hardware.

The key in the dictionary is the name of the interface. The value is all object paths to hardware implementing that interface.

7.148.2.3 getName()

```
std::string saftlib::TimingReceiver_Proxy::getName ( )
```

The logical name with which the device was connected.

Returns

The logical name with which the device was connected.

7.148.2.4 InjectEvent()

```
void saftlib::TimingReceiver_Proxy::InjectEvent (
            uint64_t event,
            uint64_t param,
            saftlib::Time time )
```

Simulate the receipt of a timing event.

Parameters

event	The event identifier which is matched against Conditions
param	The parameter field, whose meaning depends on the event ID.
time	The execution time for the event, added to condition offsets.

Sometimes it is useful to simulate the receipt of a timing event. This allows software to test that configured conditions lead to the desired behaviour without needing the data master to send anything.

7.148.2.5 signal_dispatch()

```
bool saftlib::TimingReceiver_Proxy::signal_dispatch (
            int interface_no,
```

```
int signal_no,
saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

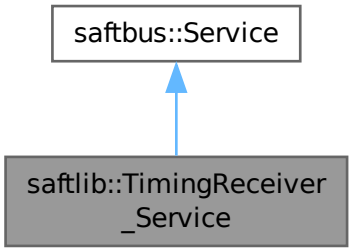
Reimplemented from saftlib::BuildIdRom_Proxy.

The documentation for this class was generated from the following files:

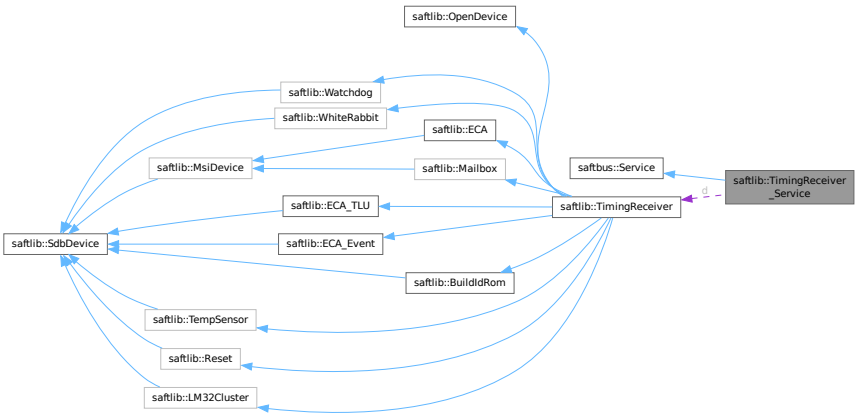
- src/TimingReceiver_Proxy.hpp
- src/TimingReceiver_Proxy.cpp

7.149 saftlib::TimingReceiver_Service Class Reference

Inheritance diagram for saftlib::TimingReceiver_Service:



Collaboration diagram for saftlib::TimingReceiver_Service:



Public Types

- typedef TimingReceiver **DriverType**

Public Member Functions

- **TimingReceiver_Service** (TimingReceiver *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Locked_dispatch_function** (bool arg_1)

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- TimingReceiver * **d**
- sigc::connection **Locked_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.149.1 Member Function Documentation

7.149.1.1 call()

```
void saftlib::TimingReceiver_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

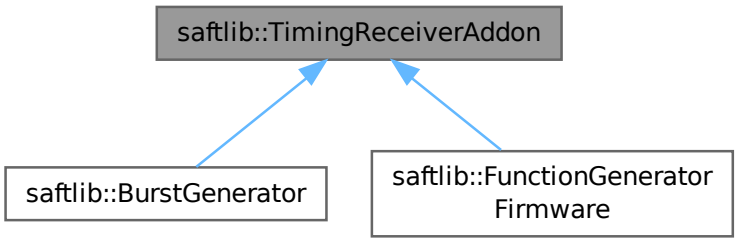
Implements saftbus::Service.

The documentation for this class was generated from the following files:

- src/TimingReceiver_Service.hpp
- src/TimingReceiver_Service.cpp

7.150 saftlib::TimingReceiverAddon Class Reference

Inheritance diagram for saftlib::TimingReceiverAddon:



Public Member Functions

- virtual std::map< std::string, std::map< std::string, std::string > > **getObjects** ()=0

The documentation for this class was generated from the following files:

- src/TimingReceiverAddon.hpp
- src/TimingReceiverAddon.cpp

7.151 saftlib::WalkEntry Struct Reference

Public Member Functions

- **WalkEntry** (int16_t n, const ECA_OpenClose &oc)

Public Attributes

- int16_t **next**
- int64_t **offset**
- uint32_t **tag**
- uint8_t **flags**
- unsigned **channel**
- unsigned **num**

The documentation for this struct was generated from the following file:

- src/ECA.cpp

7.152 software_tr::SoftwareECA::Walker Struct Reference

Public Attributes

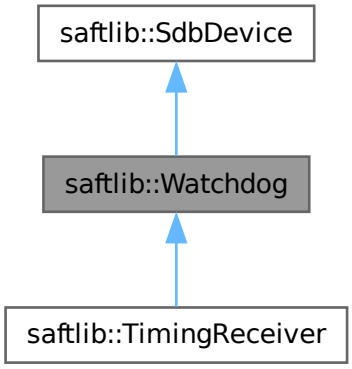
- int64_t **offset**
- int32_t **tag**
- int **next**
- int **flags**
- int **channel**
- int **num**
- bool **fired**

The documentation for this struct was generated from the following file:

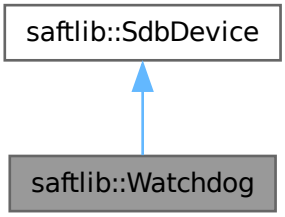
- src/saft-software-tr.cpp

7.153 saftlib::Watchdog Class Reference

Inheritance diagram for saftlib::Watchdog:



Collaboration diagram for saftlib::Watchdog:



Public Member Functions

- **Watchdog** (etherbone::Device &device)
- bool **aquire** ()
- void **update** ()

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_found=true)

Additional Inherited Members

Protected Attributes inherited from saftlib::SdbDevice

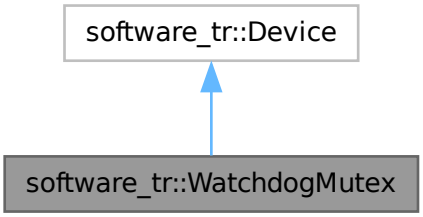
- eb_address_t **adr_first**
- etherbone::Device & **device**

The documentation for this class was generated from the following files:

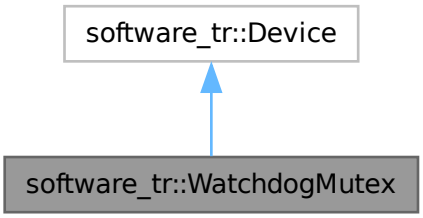
- src/Watchdog.hpp
- src/Watchdog.cpp

7.154 software_tr::WatchdogMutex Class Reference

Inheritance diagram for software_tr::WatchdogMutex:



Collaboration diagram for software_tr::WatchdogMutex:



Public Types

- enum { **vendor_id** = 0x651 , **product_id** = 0xb6232cd3 }

Public Member Functions

- **WatchdogMutex** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool read_access (uint32_t adr, int sel, uint32_t *dat_out)
- bool write_access (uint32_t adr, int sel, uint32_t dat)

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.154.1 Member Function Documentation

7.154.1.1 contains()

```
bool software_tr::WatchdogMutex::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.154.1.2 read_access()

```
bool software_tr::WatchdogMutex::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

7.154.1.3 write_access()

```
bool software_tr::WatchdogMutex::write_access (
            uint32_t adr,
            int sel,
            uint32_t dat )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

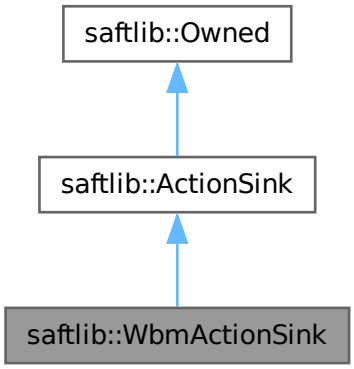
- src/saft-software-tr.cpp

7.155 saftlib::WbmActionSink Class Reference

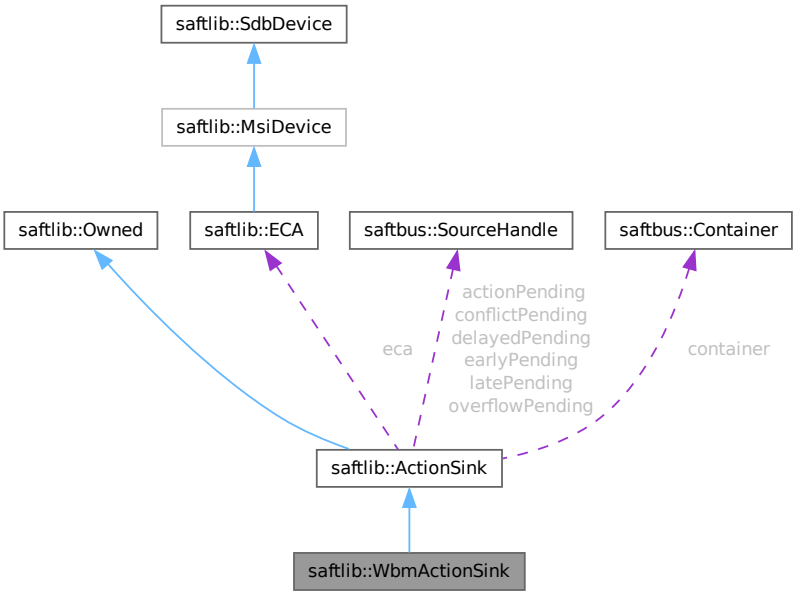
An output through which Wbm actions flow.

`#include <WbmActionSink.hpp>`

Inheritance diagram for saftlib::WbmActionSink:



Collaboration diagram for saftlib::WbmActionSink:



Public Member Functions

- **WbmActionSink** (etherbone::Device &device, ECA &eca, const std::string &object_path, const std::string &name, unsigned channel, eb_address_t acwbm_address, saftbus::Container *container=nullptr)
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag)

  Create a condition to match incoming events.
- void **ExecuteMacro** (uint32_t idx)
- void **RecordMacro** (uint32_t idx, const std::vector< std::vector< uint32_t > > &commands)
- void **ClearMacro** (uint32_t idx)
- void **ClearAllMacros** ()
- unsigned char **getStatus** () const
- uint32_t **getMaxMacros** () const
- uint32_t **getMaxSpace** () const
- bool **getEnable** () const
- uint32_t **getLastExecutedIdx** () const
- uint32_t **getLastRecordedIdx** () const
- void **setEnable** (bool val)

Public Member Functions inherited from saftlib::ActionSink

- ActionSink (ECA &eca, const std::string &action_sink_object_path, const std::string &name, unsigned channel, unsigned num, saftbus::Container *container=nullptr)

  ActionSink constructor.
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions () const

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions () const

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions () const

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset () const

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset () const

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency () const

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold () const

  Actions further into the future than this are early.
- uint16_t getCapacity () const

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull () const

  Report the largest number of pending actions seen.
- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate () const

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount () const

The number of actions lost due to Overflow.

- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount () const

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount () const

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount () const

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount () const

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount () const

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)
- void **compile** ()
- const std::string & **getObjectName** () const
- const std::string & **getObjectPath** () const
- const Conditions & **getConditions** () const
- unsigned **getChannel** () const
- unsigned **getNum** () const
- virtual void **receiveMSI** (uint8_t code)
- Condition * **getCondition** (const std::string object_path)
- void **removeCondition** (Condition *condition)
- unsigned **createConditionNumber** ()
- template<typename ConditionType , typename... Args>
  std::string **NewConditionHelper** (bool active, Args &&... args)

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Protected Attributes

- etherbone::Device & **device**
- eb_address_t **acwbm**

Protected Attributes inherited from saftlib::ActionSink

- std::string **object_path**
- ECA & **eca**
- std::string **name**
- unsigned **channel**
- unsigned **num**
- int64_t **minOffset**
- int64_t **maxOffset**
- std::chrono::nanoseconds **signalRate**
- uint64_t **overflowCount**
- uint64_t **actionCount**
- uint64_t **lateCount**
- uint64_t **earlyCount**
- uint64_t **conflictCount**
- uint64_t **delayedCount**
- std::chrono::steady_clock::time_point **overflowUpdate**
- std::chrono::steady_clock::time_point **actionUpdate**
- std::chrono::steady_clock::time_point **lateUpdate**
- std::chrono::steady_clock::time_point **earlyUpdate**
- std::chrono::steady_clock::time_point **conflictUpdate**
- std::chrono::steady_clock::time_point **delayedUpdate**
- uint64_t **latency**
- uint64_t **earlyThreshold**
- uint16_t **capacity**
- saftbus::SourceHandle **overflowPending**
- saftbus::SourceHandle **actionPending**
- saftbus::SourceHandle **latePending**
- saftbus::SourceHandle **earlyPending**
- saftbus::SourceHandle **conflictPending**
- saftbus::SourceHandle **delayedPending**
- Conditions **conditions**
- saftbus::Container * **container**

Additional Inherited Members

Public Types inherited from saftlib::ActionSink

- typedef std::map< unsigned, std::unique_ptr< Condition > > **Conditions**

Public Attributes inherited from saftlib::ActionSink

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::ActionSink

- Record **fetchError** (uint8_t code) const
- bool **updateOverflow** () const
- bool **updateAction** () const
- bool **updateLate** () const
- bool **updateEarly** () const
- bool **updateConflict** () const
- bool **updateDelayed** () const

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

7.155.1 Detailed Description

An output through which Wbm actions flow.

de.gsi.saftlib.WbmActionSink:

This inteface allows the generation of SCU timing events. A WbmActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.155.2 Member Function Documentation

7.155.2.1 NewCondition()

```
std::string saftlib::WbmActionSink::NewCondition (
             bool active,
             uint64_t id,
             uint64_t mask,
             int64_t offset,
             uint32_t tag )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
tag	The 32-bit value to send on the SCUbus
result	Object path to the created SCUbusCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ~mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SCUBUSCondition object.

The documentation for this class was generated from the following files:

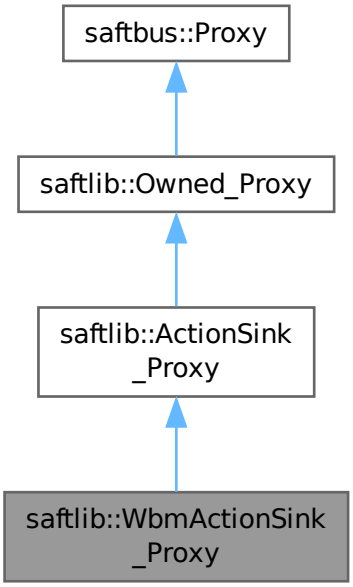
- src/WbmActionSink.hpp
- src/WbmActionSink.cpp

7.156 saftlib::WbmActionSink_Proxy Class Reference

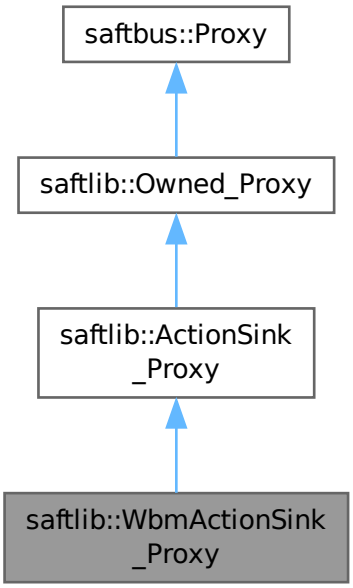
An output through which Wbm actions flow.

```
#include <WbmActionSink_Proxy.hpp>
```

Inheritance diagram for saftlib::WbmActionSink_Proxy:



Collaboration diagram for saftlib::WbmActionSink_Proxy:



Public Member Functions

- **WbmActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- std::string NewCondition (bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag)

  Create a condition to match incoming events.
- void **ExecuteMacro** (uint32_t idx)
- void **RecordMacro** (uint32_t idx, const std::vector< std::vector< uint32_t > > &commands)
- void **ClearMacro** (uint32_t idx)
- void **ClearAllMacros** ()
- unsigned char **getStatus** ()
- uint32_t **getMaxMacros** ()
- uint32_t **getMaxSpace** ()
- bool **getEnable** ()
- uint32_t **getLastExecutedIdx** ()
- uint32_t **getLastRecordedIdx** ()
- void **setEnable** (bool val)

Public Member Functions inherited from saftlib::ActionSink_Proxy

- **ActionSink_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void ToggleActive ()

  Atomically toggle the active status of conditions.
- uint16_t ReadFill ()

  Report the number of currently pending actions.
- std::vector< std::string > getAllConditions ()

  All conditions created on this ActionSink.
- std::vector< std::string > getActiveConditions ()

  All active conditions created on this ActionSink.
- std::vector< std::string > getInactiveConditions ()

  All inactive conditions created on this ActionSink.
- int64_t getMinOffset ()

  Minimum allowed offset (nanoseconds) usable in NewCondition.
- void **setMinOffset** (int64_t val)
- int64_t getMaxOffset ()

  Maximum allowed offset (nanoseconds) usable in NewCondition.
- void **setMaxOffset** (int64_t val)
- uint64_t getLatency ()

  Nanoseconds between event and earliest execution of an action.
- uint64_t getEarlyThreshold ()

  Actions further into the future than this are early.
- uint16_t getCapacity ()

  The maximum number of actions queueable without Overflow.
- uint16_t getMostFull ()

  Report the largest number of pending actions seen.

- void **setMostFull** (uint16_t val)
- uint64_t getSignalRate ()

  Minimum interval between updates (nanoseconds, default 100ms).
- void **setSignalRate** (uint64_t val)
- uint64_t getOverflowCount ()

  The number of actions lost due to Overflow.
- void **setOverflowCount** (uint64_t val)
- uint64_t getActionCount ()

  The number of actions processed by the Sink.
- void **setActionCount** (uint64_t val)
- uint64_t getLateCount ()

  The number of actions delivered late.
- void **setLateCount** (uint64_t val)
- uint64_t getEarlyCount ()

  The number of actions delivered early.
- void **setEarlyCount** (uint64_t val)
- uint64_t getConflictCount ()

  The number of actions which conflicted.
- void **setConflictCount** (uint64_t val)
- uint64_t getDelayedCount ()

  The number of actions which have been delayed.
- void **setDelayedCount** (uint64_t val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< WbmActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::ActionSink_Proxy

- static std::shared_ptr< ActionSink_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::ActionSink_Proxy

- sigc::signal< void, uint64_t > **OverflowCount**
- sigc::signal< void, uint64_t > **ActionCount**
- sigc::signal< void, uint64_t > **LateCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigLate

  : An example of a late action since last LateCount change.
- sigc::signal< void, uint64_t > **EarlyCount**
- sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigEarly

  An example of an early action since last EarlyCount change.
- sigc::signal< void, uint64_t > **ConflictCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigConflict

  An example of a conflict since last ConflictCount change.
- sigc::signal< void, uint64_t > **DelayedCount**
- sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed

  An example of a delayed action the last DelayedCount change.

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.156.1 Detailed Description

An output through which Wbm actions flow.

de.gsi.saftlib.WbmActionSink:

This inteface allows the generation of SCU timing events. A WbmActionSink is also an ActionSink and Owned.

If two SoftwareConditions are created on the same SoftwareActionSink which require simultaneous delivery of two Actions, then they will be delivered in arbitrary order, both having the 'conflict' flag set.

7.156.2 Member Function Documentation

7.156.2.1 NewCondition()

```
std::string saftlib::WbmActionSink_Proxy::NewCondition (
             bool active,
             uint64_t id,
             uint64_t mask,
             int64_t offset,
             uint32_t tag )
```

Create a condition to match incoming events.

Parameters

active	Should the condition be immediately active
id	Event ID to match incoming event IDs against
mask	Set of bits for which the event ID and id must agree
offset	Delay in nanoseconds between event and action
tag	The 32-bit value to send on the SCUbus
result	Object path to the created SCUbusCondition

This method creates a new condition that matches events whose identifier lies in the range [id & mask, id | ∼mask]. The offset acts as a delay which is added to the event's execution timestamp to determine the timestamp when the matching condition fires its action. The returned object path is a SCUBUSCondition object.

7.156.2.2 signal_dispatch()

```
bool saftlib::WbmActionSink_Proxy::signal_dispatch (
             int interface_no,
             int signal_no,
             saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

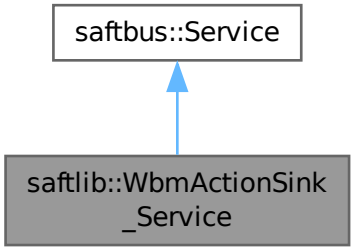
Reimplemented from saftlib::ActionSink_Proxy.

The documentation for this class was generated from the following files:

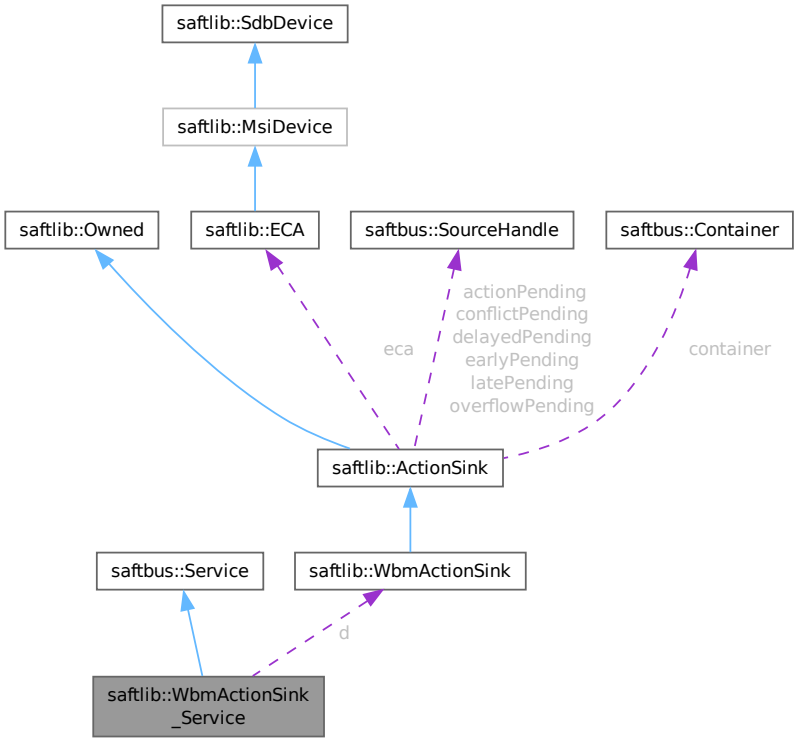
- src/WbmActionSink_Proxy.hpp
- src/WbmActionSink_Proxy.cpp

7.157 saftlib::WbmActionSink_Service Class Reference

Inheritance diagram for saftlib::WbmActionSink_Service:



Collaboration diagram for saftlib::WbmActionSink_Service:



Public Types

- typedef WbmActionSink **DriverType**

Public Member Functions

- **WbmActionSink_Service** (WbmActionSink *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **OverflowCount_dispatch_function** (uint64_t arg_1)
- void **ActionCount_dispatch_function** (uint64_t arg_1)
- void **LateCount_dispatch_function** (uint64_t arg_1)
- void **SigLate_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **EarlyCount_dispatch_function** (uint64_t arg_1)
- void **SigEarly_dispatch_function** (uint32_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **ConflictCount_dispatch_function** (uint64_t arg_1)
- void **SigConflict_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **DelayedCount_dispatch_function** (uint64_t arg_1)
- void **SigDelayed_dispatch_function** (uint64_t arg_1, uint64_t arg_2, uint64_t arg_3, saftlib::Time arg_4, saftlib::Time arg_5)
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- WbmActionSink * **d**
- sigc::connection **OverflowCount_connection**
- sigc::connection **ActionCount_connection**
- sigc::connection **LateCount_connection**
- sigc::connection **SigLate_connection**
- sigc::connection **EarlyCount_connection**
- sigc::connection **SigEarly_connection**
- sigc::connection **ConflictCount_connection**
- sigc::connection **SigConflict_connection**
- sigc::connection **DelayedCount_connection**
- sigc::connection **SigDelayed_connection**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.157.1 Member Function Documentation

7.157.1.1 call()

```
void saftlib::WbmActionSink_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

Implements saftbus::Service.

The documentation for this class was generated from the following files:

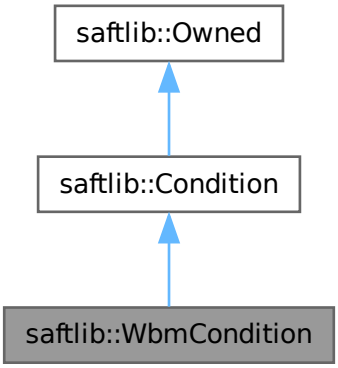
- src/WbmActionSink_Service.hpp
- src/WbmActionSink_Service.cpp

7.158 saftlib::WbmCondition Class Reference

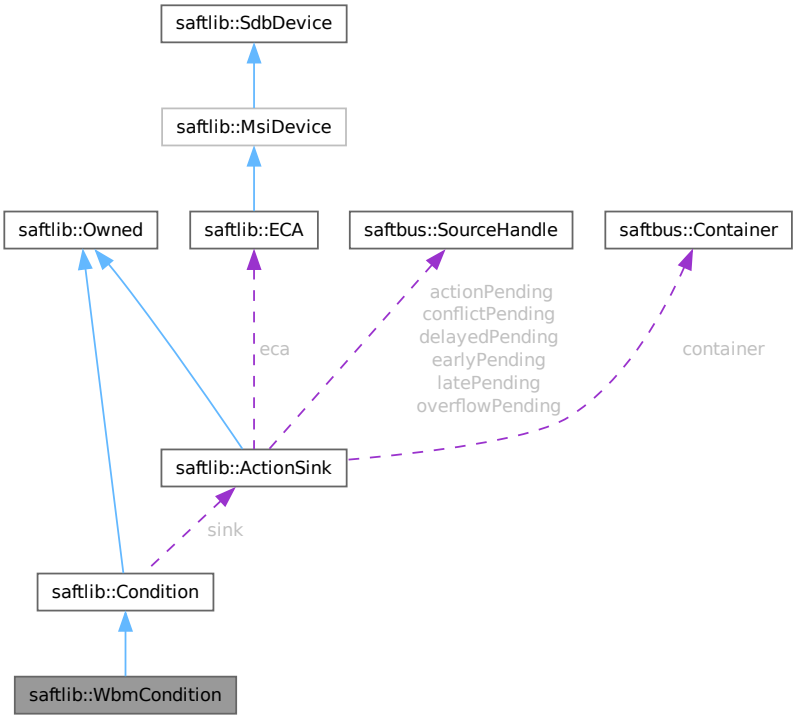
Matched against incoming events on a WbmActionSink.

```
#include <WbmCondition.hpp>
```

Inheritance diagram for saftlib::WbmCondition:



Collaboration diagram for saftlib::WbmCondition:



Public Types

- typedef WbmCondition_Service **ServiceType**

Public Member Functions

- **WbmCondition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)

Public Member Functions inherited from saftlib::Condition

- **Condition** (ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t tag, saftbus::Container *container=nullptr)
- uint64_t getID () const

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask () const

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset () const

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate () const

  Should late actions be executed? Defaults to false -->
- void **setAcceptLate** (bool val)
- bool getAcceptEarly () const

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict () const

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed () const

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive () const

  The condition should be actively matched against events.
- void **setActive** (bool val)
- std::string & **getObjectPath** ()
- uint32_t **getRawTag** () const
- void **setRawActive** (bool val)
- unsigned **getNumber** () const

Public Member Functions inherited from saftlib::Owned

- **Owned** (saftbus::Container *container)
- void **set_service** (saftbus::Service *service)

  This class only works if it has access to a service object. Service object are created after Driver class object. In order to create a functional Owned object, the service object pointer must be passed to using this function;.
- void **release_service** ()

  if a service of an Owned object is destroyed, this method must be passed as destruction callback (or must be called in the destruction callback)
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.

- std::string getOwner () const

  The client which owns this object.
- bool getDestructible () const

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Additional Inherited Members

Public Attributes inherited from saftlib::Owned

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftlib::Owned

- void **ownerOnly** () const

  Throw an exception if the caller is not the owner.

Protected Attributes inherited from saftlib::Condition

- std::string **objectPath**
- ActionSink * **sink**
- unsigned **number**
- uint64_t **id**
- uint64_t **mask**
- int64_t **offset**
- uint32_t **tag**
- bool **acceptLate**
- bool **acceptEarly**
- bool **acceptConflict**
- bool **acceptDelayed**
- bool **active**

7.158.1 Detailed Description

Matched against incoming events on a WbmActionSink.

de.gsi.saftlib.WbmCondition:

WbmConditions are created by WbmActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

The documentation for this class was generated from the following files:

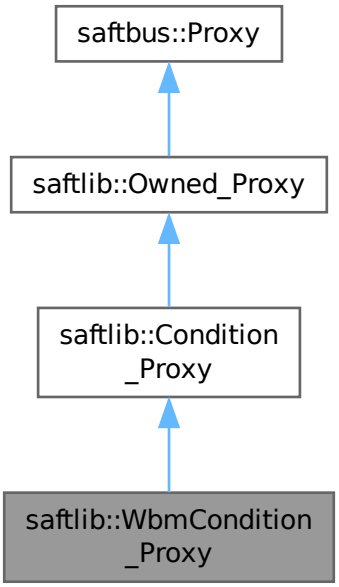
- src/WbmCondition.hpp
- src/WbmCondition.cpp

7.159 saftlib::WbmCondition_Proxy Class Reference

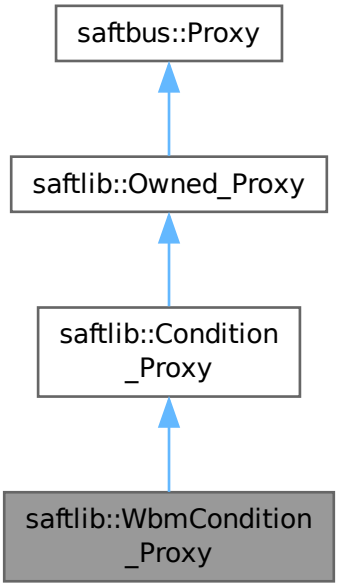
Matched against incoming events on a WbmActionSink.

`#include <WbmCondition_Proxy.hpp>`

Inheritance diagram for saftlib::WbmCondition_Proxy:



Collaboration diagram for saftlib::WbmCondition_Proxy:



Public Member Functions

- **WbmCondition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::↩ SignalGroup::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Public Member Functions inherited from saftlib::Condition_Proxy

- **Condition_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- uint64_t getID ()

  The event identifier which this condition matches against.
- void **setID** (uint64_t val)
- uint64_t getMask ()

  The mask used when comparing event IDs.
- void **setMask** (uint64_t val)
- int64_t getOffset ()

  Added to an event's time to calculate the action's time.
- void **setOffset** (int64_t val)
- bool getAcceptLate ()

Should late actions be executed? Defaults to false -->

- void **setAcceptLate** (bool val)
- bool getAcceptEarly ()

  Should early actions be executed? Defaults to false.
- void **setAcceptEarly** (bool val)
- bool getAcceptConflict ()

  Should conflicting actions be executed? Defaults to false.
- void **setAcceptConflict** (bool val)
- bool getAcceptDelayed ()

  Should delayed actions be executed? Defaults to true.
- void **setAcceptDelayed** (bool val)
- bool getActive ()

  The condition should be actively matched against events.
- void **setActive** (bool val)

Public Member Functions inherited from saftlib::Owned_Proxy

- **Owned_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup↩ ::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- void Disown ()

  Release ownership of the object.
- void Own ()

  Claim ownership of the object.
- std::string getOwner ()

  The client which owns this object.
- bool getDestructible ()

  Can the object be destroyed.
- void Destroy ()

  Destroy this object.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< WbmCondition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Condition_Proxy

- static std::shared_ptr< Condition_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Static Public Member Functions inherited from saftlib::Owned_Proxy

- static std::shared_ptr< Owned_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Additional Inherited Members

Public Attributes inherited from saftlib::Owned_Proxy

- sigc::signal< void > Destroyed

  The object was destroyed.

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.159.1 Detailed Description

Matched against incoming events on a WbmActionSink.

de.gsi.saftlib.WbmCondition:

WbmConditions are created by WbmActionSinks to select which events should generate callbacks. This interface always implies that the object also implements the general Condition interface.

7.159.2 Member Function Documentation

7.159.2.1 signal_dispatch()

```
bool saftlib::WbmCondition_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

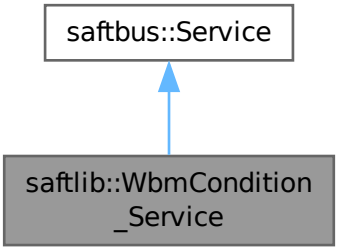
Reimplemented from saftlib::Condition_Proxy.

The documentation for this class was generated from the following files:

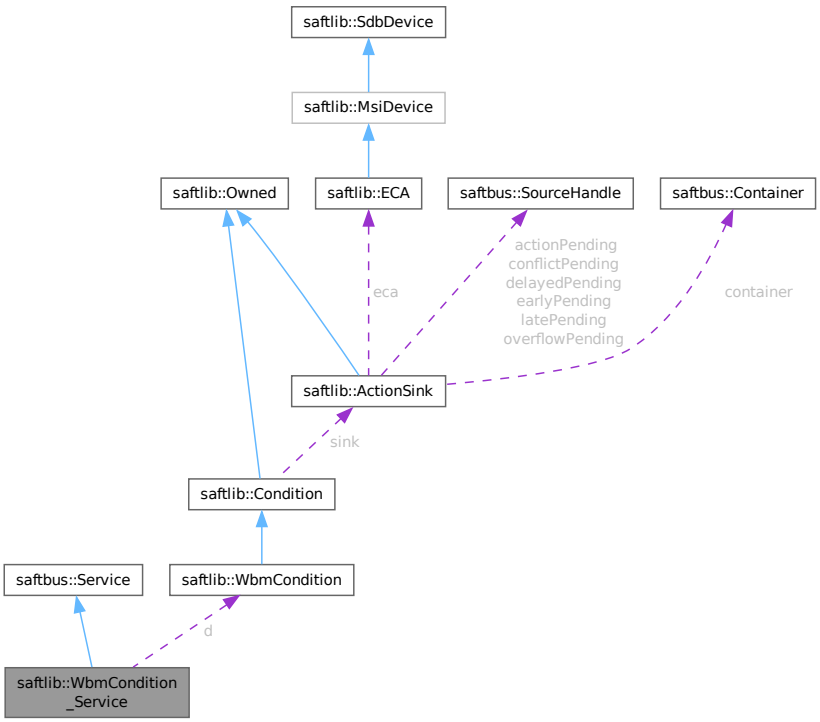
- src/WbmCondition_Proxy.hpp
- src/WbmCondition_Proxy.cpp

7.160 saftlib::WbmCondition_Service Class Reference

Inheritance diagram for saftlib::WbmCondition_Service:



Collaboration diagram for saftlib::WbmCondition_Service:



Public Types

- typedef WbmCondition **DriverType**

Public Member Functions

- **WbmCondition_Service** (WbmCondition *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Destroyed_dispatch_function** ()

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- WbmCondition * **d**
- sigc::connection **Destroyed_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.160.1 Member Function Documentation

7.160.1.1 call()

```
void saftlib::WbmCondition_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

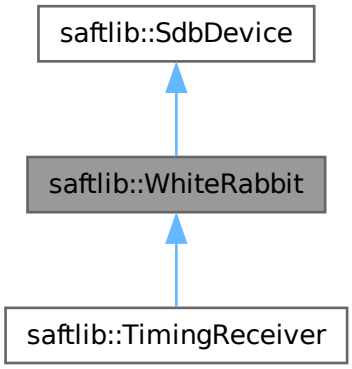
Implements saftbus::Service.

The documentation for this class was generated from the following files:

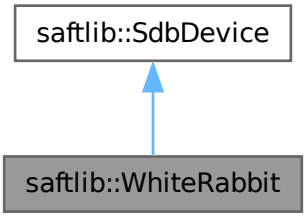
- src/WbmCondition_Service.hpp
- src/WbmCondition_Service.cpp

7.161 saftlib::WhiteRabbit Class Reference

Inheritance diagram for saftlib::WhiteRabbit:



Collaboration diagram for saftlib::WhiteRabbit:



Public Member Functions

- **WhiteRabbit** (etherbone::Device &device)
- bool getLocked () const

  The timing receiver is locked to the timing grandmaster.

Public Member Functions inherited from saftlib::SdbDevice

- **SdbDevice** (etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool throw_if_not_↩ found=true)

Public Attributes

- sigc::signal< void, bool > **Locked**

Protected Attributes

- bool **locked**

Protected Attributes inherited from saftlib::SdbDevice

- eb_address_t **adr_first**
- etherbone::Device & **device**

7.161.1 Member Function Documentation

7.161.1.1 getLocked()

```
bool saftlib::WhiteRabbit::getLocked ( ) const
```

The timing receiver is locked to the timing grandmaster.

Returns

The timing receiver is locked to the timing grandmaster.

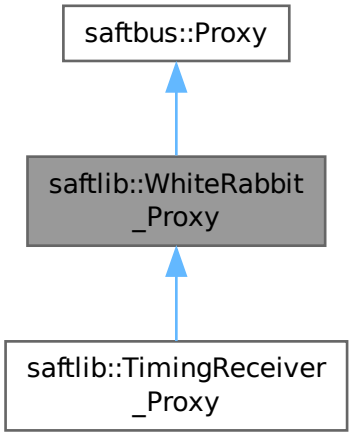
Upon power-up it takes approximately one minute until the timing receiver has a correct timestamp.

The documentation for this class was generated from the following files:

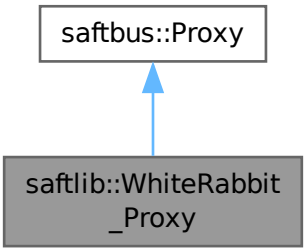
- src/WhiteRabbit.hpp
- src/WhiteRabbit.cpp

7.162 saftlib::WhiteRabbit_Proxy Class Reference

Inheritance diagram for saftlib::WhiteRabbit_Proxy:



Collaboration diagram for saftlib::WhiteRabbit_Proxy:



Public Member Functions

- **WhiteRabbit_Proxy** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::Signal↩ Group::get_global(), const std::vector< std::string > &interface_names=gen_interface_names())
- bool signal_dispatch (int interface_no, int signal_no, saftbus::Deserializer &signal_content)

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- bool getLocked ()

  The timing receiver is locked to the timing grandmaster.

Public Member Functions inherited from saftbus::Proxy

- virtual bool signal_dispatch (int interface_no, int signal_no, Deserializer &signal_content)=0

  dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no
- SignalGroup & get_signal_group ()

  The signal group to which this proxy belongs.

Static Public Member Functions

- static std::shared_ptr< WhiteRabbit_Proxy > **create** (const std::string &object_path, saftbus::SignalGroup &signal_group=saftbus::SignalGroup::get_global())

Public Attributes

- sigc::signal< void, bool > **Locked**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Proxy

- **Proxy** (const std::string &object_path, SignalGroup &signal_group, const std::vector< std::string > &interface_names)
- Serializer & get_send ()

  Get the serializer that can be used to send data to the Service object.
- Deserializer & get_received ()

  Get the deserializer that can be used to receive data from the Service object.
- int get_saftbus_object_id ()

  The id that was assigned to the Service object of this Proxy.
- std::mutex & get_client_socket_mutex ()

  the client socket is a shared resource, it should be locked before using it
- std::mutex & get_proxy_mutex ()

  each Proxy is a shared resource (mainly the serialization and deserialization buffers) and must be locked before being used
- int interface_no_from_name (const std::string &interface_name)

  needs to be called by derived classes in order to determine which interface_no they refer to.

Static Protected Member Functions inherited from saftbus::Proxy

- static ClientConnection & get_connection ()

  Get the client connection. Open the connection if that didn't happen before.

7.162.1 Member Function Documentation

7.162.1.1 getLocked()

```
bool saftlib::WhiteRabbit_Proxy::getLocked ( )
```

The timing receiver is locked to the timing grandmaster.

Returns

The timing receiver is locked to the timing grandmaster.

Upon power-up it takes approximately one minute until the timing receiver has a correct timestamp.

7.162.1.2 signal_dispatch()

```
bool saftlib::WhiteRabbit_Proxy::signal_dispatch (
            int interface_no,
            int signal_no,
            saftbus::Deserializer & signal_content )  [virtual]
```

dispatching function which triggers the actual signals (sigc::signal or std::function) based on the interface_no and signal_no

Parameters

interface_no	refers to the interface (the mapping can be obtained by interface_no_from_name).
signal_no	refers to the signal of a given interface. Signals are numbered by their appearance in the source code.

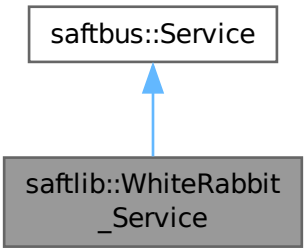
Implements saftbus::Proxy.

The documentation for this class was generated from the following files:

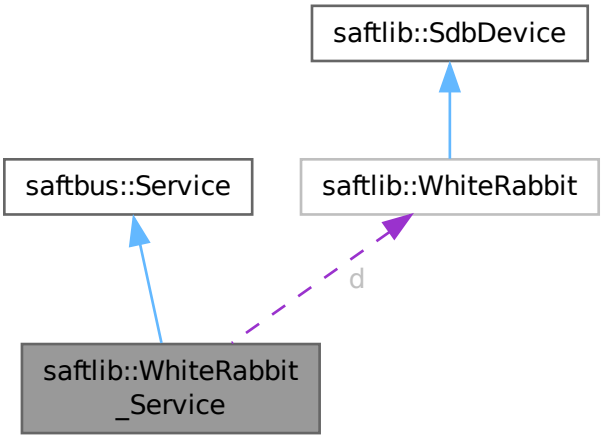
- src/WhiteRabbit_Proxy.hpp
- src/WhiteRabbit_Proxy.cpp

7.163 saftlib::WhiteRabbit_Service Class Reference

Inheritance diagram for saftlib::WhiteRabbit_Service:



Collaboration diagram for saftlib::WhiteRabbit_Service:



Public Types

- typedef WhiteRabbit **DriverType**

Public Member Functions

- **WhiteRabbit_Service** (WhiteRabbit *instance, std::function< void()> destruction_callback=std::function< void()>(), bool destoy_if_owner_quits=true)
- void call (unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer &received, saftbus::Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- void **Locked_dispatch_function** (bool arg_1)

Public Member Functions inherited from saftbus::Service

- Service (const std::vector< std::string > &interface_names, std::function< void()> destruction_↩ callback=std::function< void()>(), bool destoy_if_owner_quits=true)

  construct a Service that can be inserted into a saftbus::Container
- bool get_interface_name2no_map (const std::vector< std::string > &interface_names, std::map< std::string, int > &interface_name2no_map)

  obtain a lookup table for the interface names.
- void call (int client_fd, Deserializer &received, Serializer &send)

  execute one of the functions in one of the interfaces of the derived class.
- int **get_owner** ()
- bool **is_owned** ()
- void **set_owner** (int owner)
- void **release_owner** ()
- bool **has_destruction_callback** ()

Static Public Member Functions

- static std::vector< std::string > **gen_interface_names** ()

Public Attributes

- WhiteRabbit * **d**
- sigc::connection **Locked_connection**

Additional Inherited Members

Protected Member Functions inherited from saftbus::Service

- virtual void call (unsigned interface_no, unsigned function_no, int client_fd, Deserializer &received, Serializer &send)=0

  execute one of the functions in one of the interfaces of the derived class.
- void emit (Serializer &send)

  Send some serialized data to all clients (i.e. the SignalGroups connected to this Service).
- int **get_object_id** ()
- std::string & **get_object_path** ()
- std::vector< std::string > & **get_interface_names** ()

7.163.1 Member Function Documentation

7.163.1.1 call()

```
void saftlib::WhiteRabbit_Service::call (
            unsigned interface_no,
            unsigned function_no,
            int client_fd,
            saftbus::Deserializer & received,
            saftbus::Serializer & send )  [virtual]
```

execute one of the functions in one of the interfaces of the derived class.

Based on two numbers (interface_no and function_no), this function must extract the correct types out of received data, do something with it, put the resulting data into the send serializer. This works only, if the Proxy object and the Service object agree on interface_no, function_no, and the expected types and number of parameters and return values. Therfore, saftbus-gen always generates pairs of classes for each driver class: DriverX_Service and DriverX_Proxy.

Parameters

interface_no	identifies the interface name. The implementation of call must ensure that the inteface numbers are consistent with the interface_name2no_map lookup table that is generated in get_interface_name2no_map.
function_no	identifies the function name.
client_fd	the file descriptor (i.e. the unique ID) of the client that initiated the remote function call.
received	serialized data containing the arguments of the function call.
send	a serializer that takes the return values from the function call. It will be send back to the client that initiated the remote function call.

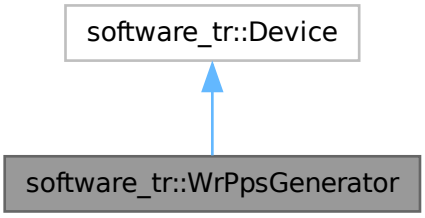
Implements saftbus::Service.

The documentation for this class was generated from the following files:

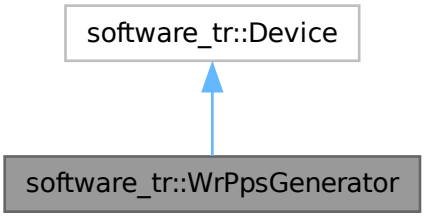
- src/WhiteRabbit_Service.hpp
- src/WhiteRabbit_Service.cpp

7.164 software_tr::WrPpsGenerator Class Reference

Inheritance diagram for software_tr::WrPpsGenerator:



Collaboration diagram for software_tr::WrPpsGenerator:



Public Types

- enum { **vendor_id** = 0xce42 , **product_id** = 0xde0d8ced }

Public Member Functions

- **WrPpsGenerator** (uint32_t adr_first, int instance)
- bool contains (uint32_t adr)
- bool read_access (uint32_t adr, int sel, uint32_t *dat_out)

Public Member Functions inherited from software_tr::Device

- virtual bool **contains** (uint32_t adr)=0
- virtual bool **read_access** (uint32_t adr, int sel, uint32_t *dat_out)
- virtual bool **write_access** (uint32_t adr, int sel, uint32_t dat)

7.164.1 Member Function Documentation

7.164.1.1 contains()

```
bool software_tr::WrPpsGenerator::contains (
            uint32_t adr )  [inline], [virtual]
```

Implements software_tr::Device.

7.164.1.2 read_access()

```
bool software_tr::WrPpsGenerator::read_access (
            uint32_t adr,
            int sel,
            uint32_t * dat_out )  [inline], [virtual]
```

Reimplemented from software_tr::Device.

The documentation for this class was generated from the following file:

- src/saft-software-tr.cpp

Chapter 8

File Documentation

8.1 chunck_allocator_rt.hpp

```
00001
00020 #ifndef SAFTBUS_CHUNCK_ALLOCATOR_RT_HPP_
00021 #define SAFTBUS_CHUNCK_ALLOCATOR_RT_HPP_
00022
00023 #include <iostream>
00024 #include <iomanip>
00025 #include <cstddef>
00026 #include <cassert>
00027 #include <cstddef>
00028
00029 template<size_t MAX_CHUNCKS, size_t CHUNCKSIZE>
00030 class ChunckAllocatorRT {
00031 public:
00032     ChunckAllocatorRT()
00033         : allocated_chuncks(0)
00034     {
00035         for (size_t i = 0; i < MAX_CHUNCKS; ++i) {
00036             chuncks[i].index = i;
00037             indices[i]       = i;
00038         }
00039     }
00040     char* malloc(size_t size) {
00041         assert(size <= CHUNCKSIZE);
00042         assert(allocated_chuncks < MAX_CHUNCKS);
00043         return chuncks[indices[allocated_chuncks++]].buffer;
00044     }
00045
00046     void free(char* ptr) {
00047         assert(allocated_chuncks > 0);
00048         assert(contains(ptr));
00049         size_t index = (ptr-chuncks[0].buffer)/sizeof(Chunck);
00050
00051         size_t &a_idx = indices[chuncks[index].index];
00052         size_t &b_idx = indices[allocated_chuncks-1];
00053
00054         size_t &a_backidx = chuncks[a_idx].index;
00055         size_t &b_backidx = chuncks[b_idx].index;
00056
00057         // std::cerr « "swapping " « a_idx « " <-> " « b_idx « std::endl;
00058         std::swap(a_idx, b_idx);
00059         // std::cerr « "swapping " « a_backidx « " <-> " « b_backidx « std::endl;
00060         std::swap(a_backidx, b_backidx);
00061
00062         --allocated_chuncks;
00063     }
00064     void print_size() {
00065         std::cerr « "filled: " « allocated_chuncks « "/" « MAX_CHUNCKS « std::endl;
00066     }
00067
00068     void print_state() {
00069         for (int i = 0; i < MAX_CHUNCKS; ++i) {
00070             std::cerr « std::setw(3) « chuncks[i].index « " ";
00071         }
00072         std::cerr « std::endl;
00073         for (int i = 0; i < MAX_CHUNCKS; ++i) {
00074             std::cerr « std::setw(3) « indices[i] « " ";
00075         }
00076         std::cerr « std::endl;
```

```
00077         for (int i = 0; i < allocated_chuncks; ++i) {
00078             std::cerr « std::setw(3) « " " « " ";
00079         }
00080         std::cerr « "  ^" « std::endl;
00081     }
00082     bool contains(char *ptr) {
00083         return (ptr >= chuncks[0].buffer
00084              && ptr <= chuncks[MAX_CHUNCKS].buffer);
00085     }
00086     bool full() {
00087         return allocated_chuncks == MAX_CHUNCKS;
00088     }
00089     bool fits(size_t n) {
00090         return n <= CHUNCKSIZE;
00091     }
00092 private:
00093     struct Chunck{
00094         char buffer[CHUNCKSIZE];
00095         size_t index;
00096     };
00097     alignas(max_align_t) Chunck chuncks[MAX_CHUNCKS];
00098     size_t indices[MAX_CHUNCKS];
00099     size_t allocated_chuncks;
00100 };
00101
00102 #endif
00103
```

8.2 client.hpp

```
00001
00021 #ifndef SAFTBUS_CLIENT_CONNECTION_HPP_
00022 #define SAFTBUS_CLIENT_CONNECTION_HPP_
00023
00024 #include "saftbus.hpp"
00025
00026 #include <cstdint>
00027 #include <memory>
00028 #include <string>
00029 #include <vector>
00030 #include <mutex>
00031 #include <algorithm>
00032
00033 #include <unistd.h>
00034
00035 namespace saftbus {
00036
00044     class ClientConnection {
00045         struct Impl; std::unique_ptr<Impl> d;
00046     friend class SignalGroup;
00047     friend class Proxy;
00048     public:
00049         ClientConnection(const std::string &socket_name = "/var/run/saftbus/saftbus");
00050         ~ClientConnection();
00056         int send(Serializer &serializer, int timeout_ms = -1);
00062         int receive(Deserializer &deserializer, int timeout_ms = -1);
00063
00070         int atomic_send_and_receive(Serializer &serializer, Deserializer &deserializer, int timeout_ms
      = -1);
00071     };
00072
00073
00074     class Proxy;
00075
00085     class SignalGroup {
00086         struct Impl; std::unique_ptr<Impl> d;
00087     friend class Proxy;
00088     public:
00089         SignalGroup();
00090         ~SignalGroup();
00091
00093         int register_proxy(Proxy *proxy);
00095         void unregister_proxy(Proxy *proxy);
00096
00100         int get_fd(); // this can be used to hook the SignalGroup into an event loop
00101
00107         int wait_for_signal(int timeout_ms = -1);
00113         int wait_for_one_signal(int timeout_ms = -1);
00114
00115         static SignalGroup &get_global();
00116     };
00117
00140     class Proxy {
```

```
00141         struct Impl; std::unique_ptr<Impl> d;
00142     friend class SignalGroup;
00143     public:
00144         virtual ~Proxy();
00149         virtual bool signal_dispatch(int interface_no, int signal_no, Deserializer &signal_content) =
      0;
00150
00154         SignalGroup& get_signal_group();
00155     protected:
00156         Proxy(const std::string &object_path, SignalGroup &signal_group, const
      std::vector<std::string> &interface_names);
00159         static ClientConnection& get_connection();
00162         Serializer&              get_send();
00165         Deserializer&            get_received();
00168         int                      get_saftbus_object_id();
00171         std::mutex&              get_client_socket_mutex();
00172
00175         std::mutex&              get_proxy_mutex();
00176
00177
00183         int interface_no_from_name(const std::string &interface_name);
00184     };
00185
00187     struct SaftbusInfo : public SerDesAble {
00189         struct ObjectInfo : public SerDesAble {
00190             unsigned object_id;
00191             std::string object_path;
00192             std::vector<std::string> interface_names;
00193             std::map<int, int> signal_fds_use_count;
00194             int owner;
00195             bool has_destruction_callback;
00196             bool destroy_if_owner_quits;
00198             void serialize(Serializer &ser) const {
00199                 ser.put(object_id);
00200                 ser.put(object_path);
00201                 ser.put(interface_names);
00202                 ser.put(signal_fds_use_count);
00203                 ser.put(owner);
00204                 ser.put(has_destruction_callback);
00205                 ser.put(destroy_if_owner_quits);
00206             }
00208             void deserialize(const Deserializer &des) {
00209                 des.get(object_id);
00210                 des.get(object_path);
00211                 des.get(interface_names);
00212                 des.get(signal_fds_use_count);
00213                 des.get(owner);
00214                 des.get(has_destruction_callback);
00215                 des.get(destroy_if_owner_quits);
00216             }
00217         };
00218         std::vector<ObjectInfo> object_infos;
00220         struct ClientInfo : public SerDesAble {
00221             pid_t process_id;
00222             int client_fd;
00223             std::map<int,int> signal_fds;
00225             void serialize(Serializer &ser) const {
00226                 ser.put(process_id);
00227                 ser.put(client_fd);
00228                 ser.put(signal_fds);
00229             }
00231             void deserialize(const Deserializer &des) {
00232                 des.get(process_id);
00233                 des.get(client_fd);
00234                 des.get(signal_fds);
00235             }
00236         };
00237         std::vector<ClientInfo> client_infos;
00238         std::vector<std::string> active_plugins;
00239         std::map<std::string, std::string> additional_info;
00241         void serialize(Serializer &ser) const {
00242             ser.put(object_infos.size());
00243             for (auto &object: object_infos) {
00244                 ser.put(object);
00245             }
00246             ser.put(client_infos.size());
00247             for (auto &client: client_infos) {
00248                 ser.put(client);
00249             }
00250             ser.put(active_plugins);
00251             ser.put(additional_info);
00252         }
00254         void deserialize(const Deserializer &des) {
00255             size_t size;
00256             des.get(size);
00257             object_infos.resize(size);
00258             for (unsigned i = 0; i < size; ++i) {
```

```
00259                 des.get(object_infos[i]);
00260             }
00261             des.get(size);
00262             client_infos.resize(size);
00263             for (unsigned i = 0; i < size; ++i) {
00264                 des.get(client_infos[i]);
00265             }
00266             des.get(active_plugins);
00267             des.get(additional_info);
00268         }
00269     };
00270 
00271 
00280     class Container_Proxy : public virtual saftbus::Proxy
00281     {
00282         static std::vector<std::string> gen_interface_names();
00283     public:
00284         Container_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group, const
      std::vector<std::string> &interface_names = std::vector<std::string>());
00285         static std::shared_ptr<Container_Proxy> create(const std::string &object_path="/saftbus",
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global(), const
      std::vector<std::string> &interface_names = std::vector<std::string>());
00286         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00287         bool load_plugin(const std::string &so_filename, const std::vector<std::string> &plugin_args =
      std::vector<std::string>());
00288         bool unload_plugin(const std::string &so_filename, const std::vector<std::string> &plugin_args
      = std::vector<std::string>());
00289         bool remove_object(const std::string &object_path);
00290         void quit();
00291         SaftbusInfo get_status();
00292     private:
00293         int interface_no;
00294 
00295     };
00296 }
00297 
00298 #endif
```

8.3 configurable_chunck_allocator_rt.hpp

```
00001 #ifndef SAFTBUS_CHUNCK_ALLOCATOR_RT_HPP_
00002 #define SAFTBUS_CHUNCK_ALLOCATOR_RT_HPP_
00003 
00004 #include <iostream>
00005 #include <iomanip>
00006 #include <string>
00007 #include <cstddef>
00008 #include <cassert>
00009 #include <cstdlib>
00010 #include <cstdio>
00011 
00012 namespace saftbus
00013 {
00014 
00015 class ChunckAllocatorRT {
00016     friend class Allocator;
00017 public:
00018     ChunckAllocatorRT(size_t max_chuncks, size_t chuncksize);
00019     ~ChunckAllocatorRT();
00020     char* malloc(size_t size);
00021     void free(char* ptr);
00022     void print_size();
00023     void print_state();
00024     bool contains(char *ptr);
00025     bool full();
00026     bool fits(size_t n);
00027 private:
00028     const size_t MAX_CHUNCKS;
00029     const size_t CHUNCKSIZE;
00030     char   *chuncks;
00031     size_t *backindices;
00032     size_t *indices;
00033     size_t allocated_chuncks;
00034 };
00035 
00036 class Allocator {
00037 public:
00038     Allocator();
00039     char* malloc(size_t n);
00040     void free(char *ptr);
00041     std::string fillstate();
00042 private:
00043     size_t num_allocators;
```

```
00044     ChunckAllocatorRT **allocators;
00045     size_t heap_allocations;
00046 };
00047
00048 // Allocator *get_allocator();
00049 // void *operator new(std::size_t n) {
00050 //   return get_allocator()->malloc(n);
00051 // }
00052 // void operator delete(void *p) {
00053 //   char *ptr = reinterpret_cast<char*>(p);
00054 //   get_allocator()->free(ptr);
00055 // }
00056
00057 }
00058 #endif
00059
```

8.4 error.hpp

```
00001
00021 #ifndef SAFTBUS_ERROR_H_
00022 #define SAFTBUS_ERROR_H_
00023
00024 #include <stdexcept>
00025 #include <string>
00026
00027 namespace saftbus
00028 {
00029
00030     class Error : public std::runtime_error
00031     {
00032     public:
00033         enum Type
00034         {
00035             NO_ERROR,
00036             INVALID_ARGS,
00037             UNKNOWN_METHOD,
00038             IO_ERROR,
00039             ACCESS_DENIED,
00040             FAILED,
00041         };
00042
00043         Error();
00044         Error(Type type, const std::string &msg);
00045         Error(const std::string &msg);
00046         Error(const Error& error);
00047
00048         Type type() const;
00049
00050     private:
00051         Type type_;
00052     };
00053
00054 }
00055
00056
00057 #endif
```

8.5 global_allocator.hpp

```
00001
00020 #ifndef SAFTBUS_GLOBAL_ALLOCATOR_HPP_
00021 #define SAFTBUS_GLOBAL_ALLOCATOR_HPP_
00022
00023 #include "chunck_allocator_rt.hpp"
00024 #include <iostream>
00025 #include <sstream>
00026
00027 class Allocator {
00028 public:
00029     Allocator() {
00030         allocator_1 = new(::malloc(sizeof(ChunckAllocatorRT<16384,128>)))
      ChunckAllocatorRT<16384,128>;
00031         allocator_2 = new(::malloc(sizeof(ChunckAllocatorRT<128,1024>))) ChunckAllocatorRT<128,1024>;
00032         allocator_3 = new(::malloc(sizeof(ChunckAllocatorRT<64,16384>))) ChunckAllocatorRT<64,16384>;
00033     }
00034     ~Allocator() {
00035         ::free(allocator_3);
00036         ::free(allocator_2);
```

```
00037         ::free(allocator_1);
00038     }
00039     char* malloc(size_t n) {
00040         // std::cerr « "--------malloc----------" « std::endl;
00041         // allocator_1->print_size();
00042         // allocator_2->print_size();
00043         // allocator_3->print_size();
00044                     if (allocator_1->fits(n) && !allocator_1->full()) {
00045             return allocator_1->malloc(n);
00046         } else if (allocator_2->fits(n) && !allocator_2->full()) {
00047             return allocator_2->malloc(n);
00048         } else if (allocator_3->fits(n) && !allocator_3->full()) {
00049             return allocator_3->malloc(n);
00050         } else {
00051             // std::cerr « "HEAP!!!!!!!!!!!!!!!!!!! " « n « std::endl;
00052             return reinterpret_cast<char*>(::malloc(n));
00053         }
00054     }
00055     void free(char *ptr) {
00056                     if (allocator_1->contains(ptr)) {
00057             allocator_1->free(ptr);
00058         } else if (allocator_2->contains(ptr)) {
00059             allocator_2->free(ptr);
00060         } else if (allocator_3->contains(ptr)) {
00061             allocator_3->free(ptr);
00062         } else {
00063             ::free(ptr);
00064         }
00065         // std::cerr « "--------free------------" « std::endl;
00066         // allocator_1->print_size();
00067         // allocator_2->print_size();
00068         // allocator_3->print_size();
00069     }
00070 private:
00071     ChunckAllocatorRT<16384,128> *allocator_1;
00072     ChunckAllocatorRT<128,1024> *allocator_2;
00073     ChunckAllocatorRT<64,16384> *allocator_3;
00074 };
00075
00076
00077 static Allocator *get_allocator() {
00078     static Allocator *allocator = new(::malloc(sizeof(*allocator))) Allocator;
00079     return allocator;
00080 }
00081 void *operator new(std::size_t n) {
00082     return get_allocator()->malloc(n);
00083 }
00084 void operator delete(void *p) {
00085     char *ptr = reinterpret_cast<char*>(p);
00086     get_allocator()->free(ptr);
00087 }
00088
00089 #endif
00090
```

8.6 loop.hpp

```
00001
00021 #ifndef SAFTBUS_LOOP_HPP_
00022 #define SAFTBUS_LOOP_HPP_
00023
00024 #include <memory>
00025 #include <iostream>
00026 #include <chrono>
00027 #include <functional>
00028 #include <vector>
00029 #include <set>
00030
00031 #include <poll.h>
00032
00033 namespace saftbus {
00034
00035     class Loop;
00037     class Source {
00038         // struct Impl; std::unique_ptr<Impl> d;
00039     friend class Loop;
00040     public:
00041         Source();
00042         virtual ~Source();
00043         virtual bool prepare(std::chrono::milliseconds &timeout_ms) = 0;
00044         virtual bool check() = 0;
00045         virtual bool dispatch() = 0;
00046         virtual std::string type() = 0;
```

```
00047         long get_id();
00048     protected:
00049         void add_poll(pollfd *pfd);
00050         void remove_poll(pollfd *pfd);
00051         void clear_poll();
00052     private:
00053         Loop *loop;
00054         std::vector<pollfd*> pfds;
00055         static long id_counter;
00056         long id;
00057     };
00059     class SourceHandle {
00060         friend class Loop;
00061         long source_id;
00062         long loop_id;
00063     public:
00064         SourceHandle() :source_id(0), loop_id(0) {}
00065         long get_source_id() const {return source_id;}
00066         long get_loop_id()   const {return loop_id;}
00067         bool connected()     const {return loop_id!=0;}
00068     };
00069
00078     class Loop {
00079         struct Impl; std::unique_ptr<Impl> d;
00080     public:
00081         Loop();
00082         ~Loop();
00083         bool iteration(bool may_block);
00084         void run();
00085         bool quit();
00086         bool quit_in(std::chrono::milliseconds wait_ms);
00087         SourceHandle connect(std::unique_ptr<Source> source);
00088
00089         template<typename T, typename... Args> // T must be subclass of Source
00090         SourceHandle connect(Args&&... args) {
00091             return connect(std::move(std::unique_ptr<T>(new T(std::forward<Args>(args)...))));
00092         }
00093         void remove(SourceHandle s);
00094         void clear(); // remove all sources
00095         static Loop &get_default();
00096     };
00097
00099     // Define two useful Source types
00101
00105     class TimeoutSource : public Source {
00106     public:
00110         TimeoutSource(std::function<bool(void)> slot, std::chrono::milliseconds interval,
      std::chrono::milliseconds offset);
00113         TimeoutSource(std::function<bool(void)> slot, std::chrono::milliseconds interval);
00114         ~TimeoutSource();
00115         bool prepare(std::chrono::milliseconds &timeout_ms) override;
00116         bool check() override;
00117         bool dispatch() override;
00118         std::string type() override;
00119     private:
00120         std::function<bool(void)> slot;
00121         std::chrono::milliseconds interval;
00122         std::chrono::time_point<std::chrono::steady_clock> dispatch_time;
00123     };
00124
00126
00132     class IoSource : public Source {
00133         friend class Loop;
00134     public:
00135         IoSource(std::function<bool(int, int)> slot, int fd, int condition);
00136         ~IoSource();
00137         bool prepare(std::chrono::milliseconds &timeout_ms) override;
00138         bool check() override;
00139         bool dispatch() override;
00140         std::string type() override;
00141     private:
00142         std::function<bool(int, int)> slot;
00143         pollfd pfd;
00144     };
00145
00146 }
00147
00148
00149 #endif
```

8.7 plugins.hpp

```
00001
```

```
00021 #ifndef SAFTBUS_PLUGINS_HPP_
00022 #define SAFTBUS_PLUGINS_HPP_
00023
00024 #include "service.hpp"
00025 #include "loop.hpp"
00026
00027 #include <string>
00028 #include <vector>
00029 #include <map>
00030
00031 namespace saftbus {
00032
00039     class LibraryLoader {
00040         struct Impl; std::unique_ptr<Impl> d;
00041     public:
00042         LibraryLoader(const std::string &so_filename);
00043         ~LibraryLoader();
00050         void create_services(Container *container, const std::vector<std::string> &args =
      std::vector<std::string>());
00051     };
00052
00053 }
00054
00055
00056
00057 #endif
```

8.8 saftbus.hpp

```
00001 /*  Copyright (C) 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Michael Reese <m.reese@gsi.de>
00004  *
00005  ********************************************************************************
00006  *  This library is free software; you can redistribute it and/or
00007  *  modify it under the terms of the GNU Lesser General Public
00008  *  License as published by the Free Software Foundation; either
00009  *  version 3 of the License, or (at your option) any later version.
00010  *
00011  *  This library is distributed in the hope that it will be useful,
00012  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  *  Lesser General Public License for more details.
00015  *
00016  *  You should have received a copy of the GNU Lesser General Public
00017  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00018  ********************************************************************************
00019  */
00020
00021 #ifndef SAFTBUS_SAFTBUS_HPP_
00022 #define SAFTBUS_SAFTBUS_HPP_
00023
00024 #include <iostream>
00025 #include <iomanip>
00026 #include <vector>
00027 #include <string>
00028 #include <map>
00029
00061 namespace saftbus {
00062
00063     enum class FunctionResult {
00064         RETURN,
00065         EXCEPTION,
00066     };
00067
00068     int write_all(int fd, const char *buffer, int size);
00069     int read_all(int fd, char *buffer, int size);
00070
00076     int sendfd(int socket, int fd);
00077
00084     int recvfd(int socket);
00085
00086     class Serializer;
00087     class Deserializer;
00088
00091     struct SerDesAble {
00092         virtual ~SerDesAble() = default;
00093         virtual void serialize(Serializer &ser) const = 0;
00094         virtual void deserialize(const Deserializer &des) = 0;
00095     };
00096
00097
00113     class Serializer
```

```
00114     {
00115     public:
00116         Serializer(int reserve = 4096)
00117         {
00118             _data.reserve(reserve);
00119             _iter = _data.begin();
00120         }
00121 
00122         // write the length of the serdes data buffer and the buffer content to file descriptor fd
00123         bool write_to(int fd);
00124         bool write_to_no_init(int fd);
00125 
00126         // this looses in overload resolution against put<SerDesAble>(const T &val)
00127         // so the wrong function is called... :(
00128         // void put(const SerDesAble &val) {
00129         //  std::cerr « "put(SerDesAble)" « std::endl;
00130         //  val.serialize(*this);
00131         // }
00132 
00133     // template <typename T>
00134     // std::enable_if_t<!std::is_base_of<serializable,T>::value> put(const T&) { std::cout « "const T&
      \n";}
00135 
00136     // template <typename T>
00137     // typename std::enable_if<std::is_base_of<serializable,T>::value>::type put(const T&) { std::cout
      « "T inherits from serializable \n";}
00138 
00139 
00140         // Types derived from SerDesAble
00141         template<typename T>
00142         typename std::enable_if<std::is_base_of<SerDesAble,T>::value>::type put(const T &val) {
00143             val.serialize(*this);
00144         }
00145 
00146         // POD struct and build-in types
00147         template<typename T>
00148         typename std::enable_if<!std::is_base_of<SerDesAble,T>::value>::type put(const T &val) {
00149             while(_data.size()%sizeof(T) != 0) _data.push_back('x'); // insert padding (reading from
      address that is not aligned to target type is undefined behavior)
00150             const char* begin = const_cast<char*>(reinterpret_cast<const char*>(&val));
00151             const char* end   = begin + sizeof(val);
00152             _data.insert(_data.end(), begin, end);
00153         }
00154 
00155         // std::vector and nested std::vector
00156         template<typename T>
00157         void put(const std::vector<T>& std_vector) {
00158             size_t size = std_vector.size();
00159             put(size);
00160             const char* begin = const_cast<char*>(reinterpret_cast<const char*>(&std_vector[0]));
00161             const char* end   = begin + size*sizeof(std_vector[0]);
00162             while(_data.size()%sizeof(T) != 0) _data.push_back('x'); // insert padding (reading from
      address that is not aligned to target type is undefined behavior)
00163             _data.insert(_data.end(), begin, end);
00164         }
00165         template<typename T>
00166         void put(const std::vector< std::vector<T, std::allocator<T> >, std::allocator< std::vector<T,
      std::allocator<T> > > >& std_vector_vector) {
00167             size_t size = std_vector_vector.size();
00168             put(size);
00169             for (size_t i = 0; i < size; ++i) {
00170                 put(std_vector_vector[i]);
00171             }
00172         }
00173         // std::string
00174         void put(const std::string& std_string) {
00175             size_t size = std_string.size();
00176             put(size);
00177             const char* begin = const_cast<char*>(reinterpret_cast<const char*>(&std_string[0]));
00178             const char* end   = begin + size*sizeof(std_string[0]);
00179             _data.insert(_data.end(), begin, end);
00180         }
00181         // std::vector<std::string>
00182         void put(const std::vector<std::string>& vector_string) {
00183             size_t size = vector_string.size();
00184             put(size);
00185             for (size_t i = 0; i < size; ++i) {
00186                 put(vector_string[i]);
00187             }
00188         }
00189         // std::map
00190         template<typename K, typename V>
00191         void put(const std::map<K,V> &std_map) {
00192             size_t size = std_map.size();
00193             put(size);
00194             for (typename std::map<K,V>::const_iterator it = std_map.begin(); it != std_map.end();
      ++it) {
```

```
00195                 put(it->first);
00196                 put(it->second);
00197             }
00198         }
00199         // // nested Serializer
00200         // void put(Serializer &ser) {
00201         //  put(ser._data);
00202         //  ser.put_init();
00203         // }
00204
00205         bool empty();
00206
00207         // has to be called before first call to put()
00208         void put_init();
00209     private:
00210
00211
00212         std::vector<char> _data;
00213         mutable std::vector<char>::const_iterator _iter;
00214     };
00215
00216
00219     class Deserializer
00220     {
00221     public:
00222         Deserializer(int reserve = 4096)
00223         {
00224             _data.reserve(reserve);
00225             _iter = _data.begin();
00226         }
00227
00228         // fill the serdes data buffer by reading data from the file descriptor fd
00229         bool read_from(int fd);
00230
00231         // Types derived from SerDesAble
00232         template<typename T>
00233         typename std::enable_if<std::is_base_of<SerDesAble,T>::value>::type // this method competed in
      overload resulution with template<typename T> get(T &val). "enable_if" lets this version win if a
      daughter class of SerDesAble is used.
00234         get(T &val) const {
00235             val.deserialize(*this);
00236         }
00237
00238         // POD struct and build-in types
00239         template<typename T>
00240         typename std::enable_if<!std::is_base_of<SerDesAble,T>::value>::type // "enable_if" excludes
      this method from the overload resolution for all tpes derived from SerDesAble.
00241         get(T &val) const {
00242             while((_iter-_data.begin())%sizeof(T) != 0) _iter+=sizeof('x'); // insert padding (reading
      from address that is not aligned to target type is undefined behavior)
00243             val    = *const_cast<T*>(reinterpret_cast<const T*>(&(*_iter)));
00244             _iter += sizeof(val);
00245         }
00246
00247         // std::vector and nested std::vector
00248         template<typename T>
00249         void get(std::vector<T> &std_vector) const {
00250             size_t size;
00251             get(size);
00252             while((_iter-_data.begin())%sizeof(T) != 0) _iter+=sizeof('x'); // insert padding (reading
      from address that is not aligned to target type is undefined behavior)
00253             const T* begin = const_cast<T*>(reinterpret_cast<const T*>(&(*_iter)));
00254             const T* end   = begin + size;
00255             std_vector.clear();
00256             std_vector.insert(std_vector.end(), begin, end);
00257             _iter += sizeof(T)*size;
00258         }
00259         template<typename T>
00260         void get(std::vector< std::vector<T, std::allocator<T> >, std::allocator< std::vector<T,
      std::allocator<T> > > >& std_vector_vector) const {
00261             size_t size;
00262             get(size);
00263             std_vector_vector.resize(size);
00264             for (size_t i = 0; i < size; ++i) {
00265                 get(std_vector_vector[i]);
00266             }
00267         }
00268         // std::string
00269         void get(std::string &std_string) const {
00270             size_t size;
00271             get(size);
00272             const char* begin = &(*_iter);
00273             const char* end   = begin + size;
00274             std_string.clear();
00275             std_string.insert(std_string.end(), begin, end);
00276             _iter += size;
00277         }
```

```
00278         // std::vector<std::string>
00279         void get(std::vector<std::string> &vector_string) const {
00280             size_t size;
00281             get(size);
00282             vector_string.resize(size);
00283             for (size_t i = 0; i < size; ++i) {
00284                 get(vector_string[i]);
00285             }
00286         }
00287         // std::map
00288         template<typename K, typename V>
00289         void get(std::map<K,V> &std_map) const {
00290             std_map.clear();
00291             size_t size;
00292             get(size);
00293             for (size_t i = 0; i < size; ++i) {
00294                 K key;
00295                 V value;
00296                 get(key);
00297                 get(value);
00298                 std_map.insert(std::make_pair(key,value));
00299             }
00300         }
00301         // // nested Deserializer
00302         // void get(Deserializer &ser) const {
00303         //  get(ser._data);
00304         //  ser.get_init();
00305         // }
00306 
00307         void save() const;
00308         void restore() const;
00309 
00310     private:
00311 
00312         // has to be called before first call to get()
00313         void get_init() const;
00314 
00315         std::vector<char> _data;
00316         mutable std::vector<char>::const_iterator _iter;
00317         mutable std::vector<char>::const_iterator _saved_iter;
00318     };
00319 
00320 
00321 
00322 
00323 }
00324 
00325 #endif
```

8.9 saftbusd-noda.cpp

```
00001 
00021 #include "loop.hpp"
00022 #include "server.hpp"
00023 #include "client.hpp"
00024 #include "service.hpp"
00025 
00026 #include <cerrno>
00027 #include <cstring>
00028 
00029 
00030 
00031 std::string print_fillstate();
00032 
00033 void usage(char *argv0) {
00034         std::cout « "saftbusd version " « VERSION « std::endl;
00035         std::cout « std::endl;
00036         std::cout « "usage: " « argv0 « " [OPTIONS] { <plugin.so> { <plugin-arg> } }" « std::endl;
00037         std::cout « std::endl;
00038         std::cout « "  <plugin.so>        is the name of a shared object files, it must have" «
      std::endl;
00039         std::cout « "                     contain a function with name \"create_services\"." «
      std::endl;
00040         std::cout « std::endl;
00041         std::cout « "  <plugin-arg>       one or more strings can be passed as arguments" « std::endl;
00042         std::cout « "                     to each plugin. They are arguments of the " « std::endl;
00043         std::cout « "                     \"create_services\" function in the shared library." «
      std::endl;
00044         std::cout « std::endl;
00045         std::cout « "options: " « std::endl;
00046         std::cout « std::endl;
00047         std::cout « " -h | --help         print this help and exit." « std::endl;
00048         std::cout « std::endl;
```

```
00049 }
00050
00051
00052 static bool saftd_already_running()
00053 {
00054   // if ClientConnection can be established, saftbus is already running
00055   try {
00056     saftbus::ClientConnection test_connection;
00057     return true;
00058   } catch (...) {
00059     return false;
00060   }
00061   return false;
00062 }
00063
00064 static bool is_int(const std::string &name) {
00065     std::istringstream in(name);
00066     unsigned i;
00067     in » i;
00068     if (!in) return false;
00069     char ch;
00070     in » ch; // nothing must follow the integer
00071     if (in) return false;
00072     return true;
00073 }
00074
00075 static bool detect_version(const std::string &name) {
00076     if (name.size() < 2) return false;
00077     if (name[0] != '.') return false;
00078     auto pos = name.substr(1).find(".");
00079     if (pos == name.npos) return is_int(name.substr(1));
00080     if (!is_int(name.substr(1,pos))) return false;
00081     return detect_version(name.substr(pos+1));
00082 }
00083
00084 static bool detect_so_file(const std::string &name) {
00085     const std::string so_ending = ".so";
00086     if (name.size() < so_ending.size()) return false;
00087     auto pos = name.find(so_ending);
00088     if (pos == name.size()-so_ending.size()) return true;
00089     if (pos == name.npos) return false;
00090     auto rest = name.substr(pos+so_ending.size()); // rest must be a version (dots and numbers) for
      example like this ".10" or ".10.0.0"
00091     if (rest[0] != '.') return false;
00092     return detect_version(rest);
00093 }
00094
00095 int main(int argc, char *argv[]) {
00096     try {
00097
00098         std::vector<std::pair<std::string, std::vector<std::string> > > plugins_and_args;
00099         for (int i = 1; i < argc; ++i) {
00100             std::string argvi(argv[i]);
00101             if (argvi == "-h" || argvi == "--help") {
00102                 usage(argv[0]);
00103                 return 0;
00104             }
00105             if (detect_so_file(argvi)) {
00106                 std::cerr « argvi « " is plugin" « std::endl;
00107                 plugins_and_args.push_back(std::make_pair(argvi, std::vector<std::string>()));
00108             } else {
00109                 std::cerr « argvi « " is argument" « std::endl;
00110                 if (plugins_and_args.empty()) {
00111                     std::cerr « "Error: no plugin specified (these are files ending with .so) before
      argument " « argvi « std::endl;
00112                     return 1;
00113                 } else {
00114                     plugins_and_args.back().second.push_back(argvi);
00115                 }
00116             }
00117         }
00118
00119         if (saftd_already_running()) {
00120             std::cerr « "Cannot start: saftbusd already running" « std::endl;
00121             return 1;
00122         }
00123
00124         saftbus::ServerConnection server_connection(plugins_and_args);
00125
00126         // add allocator fillstate as additional info to be reported by Container::get_status()
00127         if (print_fillstate().size())
      server_connection.get_container()->add_additional_info_callback("allocator", &print_fillstate);
00128
00129         saftbus::Loop::get_default().run();
00130
00131         // delete all remaining source from Loop before the plugins are unloaded
00132         saftbus::Loop::get_default().clear();
```

```
00133
00134     } catch (std::runtime_error &e) {
00135         std::cerr « "Error: " « e.what() « std::endl;
00136         saftbus::Loop::get_default().clear();
00137         return 1;
00138     }
00139
00140     return 0;
00141 }
```

8.10 server.hpp

```
00001
00021 #ifndef SAFTBUS_SERVER_CONNECTION_HPP_
00022 #define SAFTBUS_SERVER_CONNECTION_HPP_
00023
00024 #include <memory>
00025 #include <string>
00026 #include <vector>
00027 #include <map>
00028
00029 #include <unistd.h>
00030
00031 namespace saftbus {
00032
00033     class Container;
00034
00048     class ServerConnection {
00049         struct Impl; std::unique_ptr<Impl> d;
00050     public:
00051         ServerConnection(const std::vector<std::pair<std::string, std::vector<std::string> > >
      &plugins_and_args = std::vector<std::pair<std::string, std::vector<std::string> > >(),
00052                          const std::string &socket_name = "/var/run/saftbus/saftbus");
00053         ~ServerConnection();
00054
00058         void register_signal_id_for_client(int client_id, int signal_id);
00062         void unregister_signal_id_for_client(int client_id, int signal_id);
00063
00065         int get_calling_client_id();
00066
00069         Container* get_container();
00070
00071         struct ClientInfo {
00072             pid_t process_id;
00073             int client_fd;
00074             std::map<int,int> signal_fds;
00075         };
00076         std::vector<ClientInfo> get_client_info();
00077     };
00078
00079 }
00080
00081
00082 #endif
```

8.11 service.hpp

```
00001
00021 #ifndef SAFTBUS_SERVICE_HPP_
00022 #define SAFTBUS_SERVICE_HPP_
00023
00024 #include "saftbus.hpp"
00025 #include "server.hpp"
00026
00027 // for the SaftbusInfo type
00028 // @saftbus-export
00029 #include "client.hpp"
00030
00031 #include <memory>
00032 #include <string>
00033 #include <vector>
00034 #include <functional>
00035
00036 namespace saftbus {
00037
00052     class Service {
00053         struct Impl; std::unique_ptr<Impl> d;
00054         friend class Container;
00055         friend class Container_Service;
```

```
00056         friend bool operator==(std::pair<const unsigned int, std::unique_ptr<saftbus::Service> > &p,
      const int fd);
00057     public:
00067         Service(const std::vector<std::string> &interface_names, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true);
00068
00080         bool get_interface_name2no_map(const std::vector<std::string> &interface_names,
      std::map<std::string, int> &interface_name2no_map);
00081
00091         void call(int client_fd, Deserializer &received, Serializer &send);
00092         virtual ~Service();
00093
00094
00095         int get_owner();
00096         bool is_owned();
00097         void set_owner(int owner);
00098         void release_owner();
00099         bool has_destruction_callback();
00100
00101     protected:
00117         virtual void call(unsigned interface_no, unsigned function_no, int client_fd, Deserializer
      &received, Serializer &send) = 0;
00118
00119
00126         void emit(Serializer &send);
00127
00128         // @brief get the object id of this Service object in a saftbus::Container
00129         // @return the object id of this Service object in a saftbus::Container
00130         int get_object_id();
00131
00132         // @brief get the object path of this Service object in a saftbus::Container
00133         // @return the object path of this Service object in a saftbus::Container
00134         std::string &get_object_path();
00135
00136         // @brief get all interface names implemented by this Service object
00137         // @return all interface names implemented by this Service object
00138         std::vector<std::string> &get_interface_names();
00139     };
00140
00141
00142
00148     class Container {
00149         // @saftbus-default-object-path /saftbus
00150         struct Impl; std::unique_ptr<Impl> d;
00151         friend class Container_Service;
00152     public:
00153
00157         Container(ServerConnection *connection);
00158         ~Container();
00159
00161         void add_additional_info_callback(const std::string &name, std::function<std::string(void)>
      callback);
00163         void remove_additional_info_callback(const std::string &name);
00164
00170         unsigned create_object(const std::string &object_path, std::unique_ptr<Service> service);
00171
00172         Service* get_object(const std::string &object_path);
00173
00179         void destroy_service(Service *service);
00180
00181
00188         bool call_service(unsigned saftbus_object_id, int client_fd, Deserializer &received,
      Serializer &send);
00189         void remove_signal_fd(int fd);
00190
00193         void client_hung_up(int fd);
00194
00195
00198         int get_calling_client_id() const;
00199
00204         void clear();
00205
00211         Service* removal_helper(const std::string &object_path);
00212
00213
00214         // return saftbus_object_id if the object_path was found and all requested interfaces are
      implemented
00215         // return 0 if object_path was not found
00216         // return -1 if object_path was found but not all requested interfaces are implemented by the
      object
00217         // @saftbus-export
00218         int register_proxy(const std::string &object_path, const std::vector<std::string>
      interface_names, std::map<std::string,int> &interface_name2no_map, int client_fd, int
      signal_group_fd);
00219         // @saftbus-export
00220         void unregister_proxy(unsigned saftbus_object_id, int client_fd, int signal_group_fd);
00221         // @saftbus-export
```

```
00222         bool load_plugin(const std::string &so_filename, const std::vector<std::string> &args =
      std::vector<std::string>());
00223         // @saftbus-export
00224         bool unload_plugin(const std::string &so_filename, const std::vector<std::string> &args =
      std::vector<std::string>());
00225
00226
00229         // @saftbus-export
00230         bool remove_object(const std::string &object_path);
00231         // @saftbus-export
00232         void quit();
00233
00234         // @saftbus-export
00235         SaftbusInfo get_status();
00236     };
00237
00239     class Container_Service : public saftbus::Service {
00240         Container* d;
00241         static std::vector<std::string> gen_interface_names();
00242     public:
00243         typedef Container DriverType;
00244         Container_Service(Container* instance, std::function<void()> destruction_callback = nullptr );
00245         Container_Service();
00246         ~Container_Service();
00247         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00248     };
00249
00250 }
00251
00252 #endif
```

8.12 ActionSink.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef saftlib_ACTION_SINK_HPP
00023 #define saftlib_ACTION_SINK_HPP
00024
00025 #include <map>
00026 #include <set>
00027 #include <chrono>
00028 #include <vector>
00029 #include <string>
00030
00031 #include <saftbus/loop.hpp>
00032 #include <saftbus/service.hpp>
00033
00034 // @saftbus-include
00035 #include <Time.hpp>
00036 // @saftbus-include
00037 #include <sigc++/sigc++.h>
00038
00039 #include "Condition.hpp"
00040 #include "Owned.hpp"
00041 #include "ECA.hpp"
00042
00043 namespace saftlib {
00044
00045 // class ECA;
00046
00047 // de.gsi.saftlib.ActionSink:
00119 class ActionSink : public Owned
00120 {
```

```
00121     public:
00132         ActionSink(ECA &eca
00133                  , const std::string& action_sink_object_path
00134                  , const std::string& name
00135                  , unsigned channel
00136                  , unsigned num
00137                  , saftbus::Container *container = nullptr);
00138         virtual ~ActionSink();
00139 
00154         // @saftbus-export
00155         void ToggleActive();
00156 
00169         // @saftbus-export
00170         uint16_t ReadFill();
00171 
00178         // @saftbus-export
00179         std::vector< std::string > getAllConditions() const;
00180 
00188         // @saftbus-export
00189         std::vector< std::string > getActiveConditions() const;
00190 
00198         // @saftbus-export
00199         std::vector< std::string > getInactiveConditions() const;
00200 
00210         // @saftbus-export
00211         int64_t getMinOffset() const;
00212         // @saftbus-export
00213         void setMinOffset(int64_t val);
00214 
00224         // @saftbus-export
00225         int64_t getMaxOffset() const;
00226         // @saftbus-export
00227         void setMaxOffset(int64_t val);
00228 
00232         // @saftbus-export
00233         uint64_t getLatency() const;
00234 
00241         // @saftbus-export
00242         uint64_t getEarlyThreshold() const;
00243 
00254         // @saftbus-export
00255         uint16_t getCapacity() const;
00256 
00268         // @saftbus-export
00269         uint16_t getMostFull() const;
00270         // @saftbus-export
00271         void setMostFull(uint16_t val);
00272 
00281         // @saftbus-export
00282         uint64_t getSignalRate() const;
00283         // @saftbus-export
00284         void setSignalRate(uint64_t val);
00285 
00301         // @saftbus-export
00302         uint64_t getOverflowCount() const;
00303         // @saftbus-export
00304         void setOverflowCount(uint64_t val);
00305         // @saftbus-signal
00306         sigc::signal<void, uint64_t> OverflowCount;
00307         //std::function< void(uint64_t val)> OverflowCount;
00308 
00316         // @saftbus-export
00317         uint64_t getActionCount() const;
00318         // @saftbus-export
00319         void setActionCount(uint64_t val);
00320         // @saftbus-signal
00321         sigc::signal<void, uint64_t> ActionCount;
00322         //std::function< void(uint64_t val)> ActionCount;
00323 
00337         // @saftbus-export
00338         uint64_t getLateCount() const;
00339         // @saftbus-export
00340         void setLateCount(uint64_t val);
00341         // @saftbus-signal
00342         sigc::signal< void, uint64_t> LateCount;
00343         //std::function< void(uint64_t val)> LateCount;
00344 
00345 
00358         // @saftbus-signal
00359         sigc::signal<void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> SigLate;
00360         //std::function< void(uint32_t count, uint64_t event, uint64_t param, saftlib::Time deadline,
    saftlib::Time executed) > SigLate;
00361 
00362 
00363 
00377         // @saftbus-export
00378         uint64_t getEarlyCount() const;
```

```
00379         // @saftbus-export
00380         void setEarlyCount(uint64_t val);
00381         // @saftbus-signal
00382         sigc::signal<void, uint64_t> EarlyCount;
00383         // std::function< void(uint64_t val)> EarlyCount;
00384
00392         // @saftbus-signal
00393         sigc::signal< void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> SigEarly;
00394         // std::function< void(uint32_t count, uint64_t event, uint64_t param, saftlib::Time deadline,
      saftlib::Time executed) > SigEarly;
00395
00406         // @saftbus-export
00407         uint64_t getConflictCount() const;
00408         // @saftbus-export
00409         void setConflictCount(uint64_t val);
00410         // @saftbus-signal
00411         sigc::signal<void, uint64_t> ConflictCount;
00412         // std::function< void(uint64_t val)> ConflictCount;
00413
00422         // @saftbus-signal
00423         sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> SigConflict;
00424         // std::function< void(uint64_t count, uint64_t event, uint64_t param, saftlib::Time deadline,
      saftlib::Time executed) > SigConflict;
00425
00438         // @saftbus-export
00439         uint64_t getDelayedCount() const;
00440         // @saftbus-export
00441         void setDelayedCount(uint64_t val);
00442         // @saftbus-signal
00443         sigc::signal<void, uint64_t> DelayedCount;
00444         // std::function< void(uint64_t val)> DelayedCount;
00445
00446
00455         // @saftbus-signal
00456         sigc::signal< void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time > SigDelayed;
00457         // std::function< void(uint64_t count, uint64_t event, uint64_t param, saftlib::Time deadline,
      saftlib::Time executed) > SigDelayed;
00458
00459
00460         void compile();
00461
00462         // The name under which this ActionSink is listed in TimingReceiver::Iterfaces
00463         const std::string &getObjectName() const;
00464
00465         const std::string &getObjectPath() const;
00466
00467
00468         // Used by TimingReciever::compile
00469         typedef std::map<unsigned, std::unique_ptr<Condition> > Conditions;
00470         const Conditions& getConditions() const;
00471
00472         unsigned getChannel() const;
00473         unsigned getNum() const;
00474
00475         // Receive MSI from TimingReceiver
00476         virtual void receiveMSI(uint8_t code);
00477
00478
00479
00480         Condition *getCondition(const std::string object_path);
00481
00482         // Useful for Condition destroy methods
00483         void removeCondition(Condition *condition);
00484
00485         // choose a random number that is not already used by another condition
00486         unsigned createConditionNumber();
00487
00488         template<typename ConditionType, typename... Args>
00489         std::string NewConditionHelper(bool active, Args&&... args) {
00490             unsigned number = createConditionNumber();
00491             std::unique_ptr<ConditionType> condition(new ConditionType(this, number, active,
      std::forward<Args>(args)...));
00492             std::string path = condition->getObjectPath();
00493             if (container) {
00494                 std::unique_ptr<typename ConditionType::ServiceType> service(new typename
      ConditionType::ServiceType(condition.get(), std::bind(&ActionSink::removeCondition, this,
      condition.get())));
00495                 service->set_owner(container->get_calling_client_id());
00496                 condition->set_service(service.get());
00497                 container->create_object(path, std::move(service));
00498             }
00499             conditions.insert(std::make_pair(number, std::move(condition)));
00500             if (active) {
00501                 eca.compile();
00502             }
00503             return path;
00504         }
```

```
00505
00506
00507     protected:
00508         std::string object_path;
00509         ECA &eca;
00510         std::string name;
00511         unsigned channel;
00512         unsigned num;
00513
00514
00515         // User controlled values
00516         int64_t minOffset, maxOffset;
00517         std::chrono::nanoseconds signalRate;
00518
00519         // cached counters
00520         mutable uint64_t overflowCount;
00521         mutable uint64_t actionCount;
00522         mutable uint64_t lateCount;
00523         mutable uint64_t earlyCount;
00524         mutable uint64_t conflictCount;
00525         mutable uint64_t delayedCount;
00526
00527         // last update of counters (for throttled)
00528         mutable std::chrono::steady_clock::time_point overflowUpdate;
00529         mutable std::chrono::steady_clock::time_point actionUpdate;
00530         mutable std::chrono::steady_clock::time_point lateUpdate;
00531         mutable std::chrono::steady_clock::time_point earlyUpdate;
00532         mutable std::chrono::steady_clock::time_point conflictUpdate;
00533         mutable std::chrono::steady_clock::time_point delayedUpdate;
00534
00535         // constant hardware values
00536         uint64_t latency;
00537         uint64_t earlyThreshold;
00538         uint16_t capacity;
00539
00540         // pending timeouts to refresh counters
00541         saftbus::SourceHandle overflowPending;
00542         saftbus::SourceHandle actionPending;
00543         saftbus::SourceHandle latePending;
00544         saftbus::SourceHandle earlyPending;
00545         saftbus::SourceHandle conflictPending;
00546         saftbus::SourceHandle delayedPending;
00547
00548         struct Record {
00549             uint64_t event;
00550             uint64_t param;
00551             uint64_t deadline;
00552             uint64_t executed;
00553             uint64_t count;
00554         };
00555         Record fetchError(uint8_t code) const;
00556
00557         bool updateOverflow() const;
00558         bool updateAction() const;
00559         bool updateLate() const;
00560         bool updateEarly() const;
00561         bool updateConflict() const;
00562         bool updateDelayed() const;
00563
00564         // conditions must come after dev to ensure safe cleanup on ~Condition
00565         Conditions conditions;
00566
00567
00568         saftbus::Container *container;
00569 };
00570
00571 }
00572
00573 #endif
```

8.13 ActionSink_Proxy.hpp

```
00001 #ifndef ActionSink_PROXY_HPP_
00002 #define ActionSink_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006 #include <sigc++/sigc++.h>
00007
00008 #include <Time.hpp>
00009 #include <sigc++/sigc++.h>
00010
00011 #include "Owned_Proxy.hpp"
```

```
00012
00013 namespace saftlib {
00014
00015 // de.gsi.saftlib.ActionSink:
00087     class ActionSink_Proxy : public  Owned_Proxy     {
00088         static std::vector<std::string> gen_interface_names();
00089     public:
00090         ActionSink_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00091         static std::shared_ptr<ActionSink_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00092         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00107         // @saftbus-export
00108         void ToggleActive();
00121         // @saftbus-export
00122         uint16_t ReadFill();
00129         // @saftbus-export
00130         std::vector< std::string > getAllConditions();
00138         // @saftbus-export
00139         std::vector< std::string > getActiveConditions();
00147         // @saftbus-export
00148         std::vector< std::string > getInactiveConditions();
00158         // @saftbus-export
00159         int64_t getMinOffset();
00160         // @saftbus-export
00161         void setMinOffset(int64_t val);
00171         // @saftbus-export
00172         int64_t getMaxOffset();
00173         // @saftbus-export
00174         void setMaxOffset(int64_t val);
00178         // @saftbus-export
00179         uint64_t getLatency();
00186         // @saftbus-export
00187         uint64_t getEarlyThreshold();
00198         // @saftbus-export
00199         uint16_t getCapacity();
00211         // @saftbus-export
00212         uint16_t getMostFull();
00213         // @saftbus-export
00214         void setMostFull(uint16_t val);
00223         // @saftbus-export
00224         uint64_t getSignalRate();
00225         // @saftbus-export
00226         void setSignalRate(uint64_t val);
00242         // @saftbus-export
00243         uint64_t getOverflowCount();
00244         // @saftbus-export
00245         void setOverflowCount(uint64_t val);
00253         // @saftbus-export
00254         uint64_t getActionCount();
00255         // @saftbus-export
00256         void setActionCount(uint64_t val);
00270         // @saftbus-export
00271         uint64_t getLateCount();
00272         // @saftbus-export
00273         void setLateCount(uint64_t val);
00287         // @saftbus-export
00288         uint64_t getEarlyCount();
00289         // @saftbus-export
00290         void setEarlyCount(uint64_t val);
00301         // @saftbus-export
00302         uint64_t getConflictCount();
00303         // @saftbus-export
00304         void setConflictCount(uint64_t val);
00317         // @saftbus-export
00318         uint64_t getDelayedCount();
00319         // @saftbus-export
00320         void setDelayedCount(uint64_t val);
00321         // @saftbus-signal
00322         sigc::signal<void, uint64_t> OverflowCount;
00323         // @saftbus-signal
00324         sigc::signal<void, uint64_t> ActionCount;
00325         // @saftbus-signal
00326         sigc::signal<void, uint64_t> LateCount;
00339         // @saftbus-signal
00340         sigc::signal<void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> SigLate;
00341         // @saftbus-signal
00342         sigc::signal<void, uint64_t> EarlyCount;
00350         // @saftbus-signal
00351         sigc::signal<void, uint32_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> SigEarly;
00352         // @saftbus-signal
00353         sigc::signal<void, uint64_t> ConflictCount;
00362         // @saftbus-signal
00363         sigc::signal<void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> SigConflict;
00364         // @saftbus-signal
00365         sigc::signal<void, uint64_t> DelayedCount;
```

```
00374         // @saftbus-signal
00375         sigc::signal<void, uint64_t, uint64_t, uint64_t, saftlib::Time, saftlib::Time> SigDelayed;
00376     private:
00377         int interface_no;
00378 
00379     };
00380 
00381 }
00382 
00383 #endif
```

8.14 ActionSink_Service.hpp

```
00001 #ifndef ActionSink_SERVICE_HPP_
00002 #define ActionSink_SERVICE_HPP_
00003 
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006 
00007 #include <functional>
00008 
00009 namespace saftlib {
00010 
00011     class ActionSink;
00012     class ActionSink_Service : public saftbus::Service {
00013     public:
00014         ActionSink* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef ActionSink DriverType;
00017         ActionSink_Service(ActionSink* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         ActionSink_Service();
00019         ~ActionSink_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021 
00022         void OverflowCount_dispatch_function(uint64_t arg_1);
00023         sigc::connection OverflowCount_connection;
00024         void ActionCount_dispatch_function(uint64_t arg_1);
00025         sigc::connection ActionCount_connection;
00026         void LateCount_dispatch_function(uint64_t arg_1);
00027         sigc::connection LateCount_connection;
00028         void SigLate_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00029         sigc::connection SigLate_connection;
00030         void EarlyCount_dispatch_function(uint64_t arg_1);
00031         sigc::connection EarlyCount_connection;
00032         void SigEarly_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00033         sigc::connection SigEarly_connection;
00034         void ConflictCount_dispatch_function(uint64_t arg_1);
00035         sigc::connection ConflictCount_connection;
00036         void SigConflict_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00037         sigc::connection SigConflict_connection;
00038         void DelayedCount_dispatch_function(uint64_t arg_1);
00039         sigc::connection DelayedCount_connection;
00040         void SigDelayed_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00041         sigc::connection SigDelayed_connection;
00042         void Destroyed_dispatch_function();
00043         sigc::connection Destroyed_connection;
00044     };
00045 
00046 }
00047 
00048 #endif
```

8.15 ats_regs.h

```
00001 
00020 #ifndef TEMP_SENSOR_H
00021 #define TEMP_SENSOR_H
00022 
00023 #define ATS_SDB_VENDOR_ID   0x00000651  // Altera temperature sensor (ATS)
00024 #define ATS_SDB_DEVICE_ID   0x7e3d5e25
00025 
00026 #define ALTERA_TEMP_DEGREE  0x04        //Register for Altera temperature sensor (in degree)
00027 
00028 #endif
```

8.16 bg_regs.h

```
00001
00020 #ifndef BG_REGS_H
00021 #define BG_REGS_H
00022
00023 #include "burstgen_shared_mmap.h"
00024
00025 // Locate shared memory of processors
00026 #define LM32_RAM_USER_VENDOR      0x651       //vendor ID
00027 #define LM32_RAM_USER_PRODUCT     0x54111351  //product ID
00028 #define LM32_RAM_USER_VMAJOR      1           //major revision
00029 #define LM32_RAM_USER_VMINOR      1           //minor revision
00030
00031 #define LM32_CLUSTER_ROM_VENDOR   0x651
00032 #define LM32_CLUSTER_ROM_PRODUCT  0x10040086
00033
00034 #define MSI_MAILBOX_VENDOR        0x651
00035 #define MSI_MAILBOX_PRODUCT       0xfab0bdd8
00036 #define MB_SLOT_RANGE             128          // 0 .. 127
00037 #define MB_SLOT_CFG_FREE          0xffffffffUL
00038
00039 // defines
00040 #define BG_FW_ID                  0xb2b2b2b2
00041
00042 // definitions of buffers in shared memory
00043 #define SHARED_COMMON_SIZE        2048              // reserved size for common-lib registers
00044 #define SHM_BASE                  SHARED_OFFS + SHARED_COMMON_SIZE
00045
00046 #define SHM_FW_ID                 SHM_BASE          // offset to app specific section in shared
      memory
00047 #define SHM_MB_SLOT               SHM_BASE + 0x04UL  // offset to the mailbox slof for lm32
00048 #define SHM_MB_SLOT_HOST          SHM_BASE + 0x0CUL  // offset to the mailbox slot for host
00049 #define SHM_COMMON_BEGIN          SHM_BASE + 0x10UL  // start address of the common-lib section
00050 #define SHM_COMMON_END            SHM_BASE + 0x14UL  // end address of the common-lib section
00051 #define SHM_COMMON_CMD            SHM_BASE + 0x18UL  // address of the command buffer (common-lib)
00052 #define SHM_COMMON_STATE          SHM_BASE + 0x1CUL  // address of the firmware state buffer
      (common-lib)
00053 #define SHM_CMD_ARGS              SHM_BASE + 0x20UL  // offset to the command argument buffer
00054
00055 // index of the buffers in shared memory
00056 enum SHM_BUF_IDX {
00057   COMMON_BEGIN = 0,
00058   COMMON_END,
00059   COMMON_CMD,
00060   COMMON_STATE,
00061   CMD_ARGS,
00062   N_SHM_IDX
00063 };
00064
00065 #define EVT_ID_IO_H32             0x0000fca0UL  // event id of timing message for IO actions (hi32)
00066 #define EVT_ID_IO_L32             0x00000000UL  // event id of timing message for IO actions (lo32)
00067 #define EVT_MASK_IO               0xffffffffffffffffULL
00068
00069 // user commands for the burst generator
00070 #define CMD_SHOW_ALL              0x21UL         // show pulse parameters, pulse cycles
00071 #define CMD_GET_PARAM             0x22UL         // get pulse parameters
00072 #define CMD_GET_CYCLE             0x23UL         // get pulse cycles
00073 #define CMD_LS_BURST              0x24UL         // list burst (burst ids or burst info)
00074 #define CMD_MK_BURST              0x25UL         // declare new burst
00075 #define CMD_RM_BURST              0x26UL         // remove burst
00076 #define CMD_DE_BURST              0x27UL         // dis/enable burst
00077 #define CMD_RD_MSI_ECPU           0x30UL        // read and show the content of ECA MSI registers (MSI
      enable, MSI destination address)
00078 #define CMD_RD_ECPU_CHAN          0x31UL        // read and show the content of ECA counters for the
      eCPU action channel
00079 #define CMD_RD_ECPU_QUEUE         0x32UL        // read and show the content of ECA queue connected to
      the eCPU action channel
00080 #define CMD_LS_FW_ID              0x33UL        // list the firmware id (the value is written to the
      shared input buffer)
00081
00082 #define CTL_DIS                   0x0000UL
00083 #define CTL_EN                    0x0001UL
00084 #define CTL_VALID                 0x8000UL
00085
00086 #define N_BURSTS                  16            // maximum number of bursts
00087
00088 enum BURST_INFO {                               // burst info fields
00089   INFO_ID,
00090   INFO_IO_TYPE,
00091   INFO_IO_IDX,
00092   INFO_START_EVT_H32,
00093   INFO_START_EVT_L32,
00094   INFO_STOP_EVT_H32,
00095   INFO_STOP_EVT_L32,
00096   INFO_LOOPS_H32,
00097   INFO_LOOPS_L32,
```

```
00098   INFO_ACTIONS_H32,
00099   INFO_ACTIONS_L32,
00100   INFO_FLAG,
00101   N_BURST_INFO
00102 };
00103
00104 #define INTERVAL_200US            200000ULL
00105
00106 #define __BG_RETURN_SUCCESS       0
00107 #define __BG_RETURN_FAILURE       0xffffffff
00108 #define __BG_RETURN_OPTION_INVAL  0xfffffffe
00109
00110 #endif
```

8.17 build.hpp

```
00001 /*  Copyright (C) 2011-2016 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *
00005  ********************************************************************************
00006  *  This library is free software; you can redistribute it and/or
00007  *  modify it under the terms of the GNU Lesser General Public
00008  *  License as published by the Free Software Foundation; either
00009  *  version 3 of the License, or (at your option) any later version.
00010  *
00011  *  This library is distributed in the hope that it will be useful,
00012  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  *  Lesser General Public License for more details.
00015  *
00016  *  You should have received a copy of the GNU Lesser General Public
00017  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00018  ********************************************************************************
00019  */
00020 #ifndef BUILD_H
00021 #define BUILD_H
00022
00023 namespace saftlib {
00024
00025 extern const char buildInfo[];
00026 extern const char sourceVersion[];
00027
00028 }
00029
00030 #endif
```

8.18 BuildIdRom.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_BUILD_ID_ROM_HPP_
00023 #define saftlib_BUILD_ID_ROM_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
```

```
00031
00032 #include "SdbDevice.hpp"
00033
00034 #include <map>
00035 #include <string>
00036
00037 namespace saftlib {
00038
00041 class BuildIdRom : public SdbDevice {
00042     std::map<std::string, std::string> gateware_info;
00043     void setupGatewareInfo(uint32_t address);
00044 public:
00045     BuildIdRom(etherbone::Device &device);
00046
00050     // @saftbus-export
00051     std::map< std::string, std::string > getGatewareInfo() const;
00052
00056     // @saftbus-export
00057     std::string getGatewareVersion() const;
00058
00059 };
00060
00061
00062 }
00063
00064
00065 #endif
```

8.19 BuildIdRom_Proxy.hpp

```
00001 #ifndef BuildIdRom_PROXY_HPP_
00002 #define BuildIdRom_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008
00009 namespace saftlib {
00010
00013     class BuildIdRom_Proxy : public virtual saftbus::Proxy
00014     {
00015         static std::vector<std::string> gen_interface_names();
00016     public:
00017         BuildIdRom_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00018         static std::shared_ptr<BuildIdRom_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00019         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00023     // @saftbus-export
00024         std::map< std::string, std::string > getGatewareInfo();
00028     // @saftbus-export
00029         std::string getGatewareVersion();
00030     private:
00031         int interface_no;
00032
00033     };
00034
00035 }
00036
00037 #endif
```

8.20 BuildIdRom_Service.hpp

```
00001 #ifndef BuildIdRom_SERVICE_HPP_
00002 #define BuildIdRom_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class BuildIdRom;
00012     class BuildIdRom_Service : public saftbus::Service {
00013     public:
00014         BuildIdRom* d;
```

```
00015         static std::vector<std::string> gen_interface_names();
00016         typedef BuildIdRom DriverType;
00017         BuildIdRom_Service(BuildIdRom* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         BuildIdRom_Service();
00019         ~BuildIdRom_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.21 burstgen_shared_mmap.h

```
00001 #ifndef BURSTGEN_SHARED_MMAP_H
00002 #define BURSTGEN_SHARED_MMAP_H
00003 //Location of Buildid and Shared Section in LM32 Memory, to be used by host
00004
00005 #define INT_BASE_ADR        0x10000000
00006 #define RAM_SIZE        131072
00007 #define SHARED_SIZE     16K
00008 #define BUILDID_OFFS        0x100
00009 #define SHARED_OFFS     (0x100 + 0x400)
00010 #endif
```

8.22 BurstGenerator.hpp

```
00001
00002 #ifndef BURST_GENERATOR_FIRMWARE_HPP_
00003 #define BURST_GENERATOR_FIRMWARE_HPP_
00004
00005 #include "Owned.hpp"
00006 #include <TimingReceiver.hpp>
00007 #include <TimingReceiverAddon.hpp>
00008 #include <Mailbox.hpp>
00009
00010 #include <saftbus/service.hpp> // for saftbus::Container
00011
00012 #include <sigc++/sigc++.h>
00013
00014 #include <memory>
00015
00016 namespace saftlib {
00017
00024   class BurstGenerator : public Owned , public TimingReceiverAddon
00025   {
00026
00027     public:
00028       std::map<std::string, std::map<std::string, std::string> > getObjects(); // TimingReceiverAddon
      pure-virtual override
00029       std::string getObjectPath(); // used in create_service.cpp
00030
00031       BurstGenerator(saftbus::Container *container, SAFTd *saft_daemon, TimingReceiver
      *timing_receiver);
00032       ~BurstGenerator();
00033
00034
00043       // @saftbus-export
00044       int32_t instruct(uint32_t code, const std::vector< uint32_t >& args);
00045
00052       // @saftbus-export
00053       std::vector< uint32_t > readBurstInfo(uint32_t id);
00054
00060       // @saftbus-export
00061       std::vector< uint32_t > readSharedBuffer(uint32_t size);
00062
00063
00067       // @saftbus-export
00068       uint32_t readState();
00069
00070
00073       // @saftbus-export
00074       uint32_t getResponse() const;
00075
00081       // @saftbus-export
00082       sigc::signal<void, uint32_t> sigInstComplete;
```

```
00083
00084     protected:
00085       bool firmwareRunning(uint32_t id);
00086       void msi_handler(eb_data_t msg);
00087
00088       std::string                objectPath;
00089       SAFTd                      *saftd;        // in saftlib-v3 SAFTd is needed for request_irq and
      release_irq
00090       TimingReceiver             *tr;           // TimingReceiver provides all needed SdbDevices
00091       etherbone::Device&         device;        // a reference to the etherbone::Device for quick
      access
00092       std::unique_ptr<IRQ>       my_msi;        // my msi (the address of this is used to configure
      mailbox)
00093       std::unique_ptr<Mailbox::Slot> my_slot;   // mailbox slot subscribed by me
00094
00095       int                            bg_slot;   // mailbox slot-index subscribed by the burst
      generator
00096                                                 // The slot is owned by the firmware and not by the
      host-side driver.
00097                                                 // So no Mailbox::Slot object is used (it would
      release the slot on destruction)
00098                                                 // this doesn't seem to be used
00099
00100       uint32_t                   response;      // instruction result and instruction code (sent by
      MSI)
00101
00102       bool                       found_bg_fw;
00103       eb_address_t               ram_base;      // start of lm32 user ram
00104       std::vector<eb_address_t>  shm_buffer;    // app specific buffers in shared memory (for embedded
      lm32 communication)
00105
00106   };
00107
00108 }
00109
00110 #endif
```

8.23 BurstGenerator_Proxy.hpp

```
00001 #ifndef BurstGenerator_PROXY_HPP_
00002 #define BurstGenerator_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006 #include <sigc++/sigc++.h>
00007
00008
00009 #include "Owned_Proxy.hpp"
00010
00011 namespace saftlib {
00012
00019     class BurstGenerator_Proxy : public  Owned_Proxy     {
00020         static std::vector<std::string> gen_interface_names();
00021     public:
00022         BurstGenerator_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00023         static std::shared_ptr<BurstGenerator_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00024         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00033       // @saftbus-export
00034         int32_t instruct(uint32_t code, const std::vector< uint32_t >& args);
00041       // @saftbus-export
00042         std::vector< uint32_t > readBurstInfo(uint32_t id);
00048       // @saftbus-export
00049         std::vector< uint32_t > readSharedBuffer(uint32_t size);
00053       // @saftbus-export
00054         uint32_t readState();
00057       // @saftbus-export
00058         uint32_t getResponse();
00064       // @saftbus-export
00065         sigc::signal<void, uint32_t> sigInstComplete;
00066     private:
00067         int interface_no;
00068
00069     };
00070
00071 }
00072
00073 #endif
```

8.24 BurstGenerator_Service.hpp

```
00001 #ifndef BurstGenerator_SERVICE_HPP_
00002 #define BurstGenerator_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class BurstGenerator;
00012     class BurstGenerator_Service : public saftbus::Service {
00013     public:
00014         BurstGenerator* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef BurstGenerator DriverType;
00017         BurstGenerator_Service(BurstGenerator* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         BurstGenerator_Service();
00019         ~BurstGenerator_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void sigInstComplete_dispatch_function(uint32_t arg_1);
00023         sigc::connection sigInstComplete_connection;
00024         void Destroyed_dispatch_function();
00025         sigc::connection Destroyed_connection;
00026     };
00027
00028 }
00029
00030 #endif
```

8.25 CommonFunctions.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef COMMON_FUNCTIONS_H
00023 #define COMMON_FUNCTIONS_H
00024
00025 #include <iostream>
00026 #include <iomanip>
00027 #include "Time.hpp"
00028
00029 #include <saftbus/client.hpp>
00030
00031 #include <time.h>
00032 #include <sys/time.h>
00033 #include <stdio.h>
00034 #include <string.h>
00035 #include <sstream>
00036 #include <inttypes.h>
00037
00038 // modes for printing to cout
00039 const uint32_t PMODE_NONE     = 0x0;
00040 const uint32_t PMODE_DEC      = 0x1;
00041 const uint32_t PMODE_HEX      = 0x2;
00042 const uint32_t PMODE_VERBOSE  = 0x4;
00043 const uint32_t PMODE_UTC      = 0x8;
00044
00045 // formatting of mask for action sink
00046 uint64_t     tr_mask(int i                        // number of bits
```

```
00047                     );
00048
00049 // formatting of date for output
00050 std::string tr_formatDate(saftlib::Time time,     // time [ns]
00051                           uint32_t pmode,         // mode for printing
00052                           bool     json           // JSON output
00053                           );
00054
00055 // formatting of action event ID for output
00056 std::string tr_formatActionEvent(uint64_t id,       // 64bit event ID
00057                                  uint32_t pmode,    // mode for printing
00058                                  bool     json      // JSON output
00059                                  );
00060
00061 // formatting of action param for output; the format depends also on evtNo, except if evtNo == 0xFFFF
      FFFF
00062 std::string tr_formatActionParam(uint64_t param,    // 64bit parameter
00063                                  uint32_t evtNo,    // evtNo (currently 12 bit) - part of the 64 bit
      event ID
00064                                  uint32_t pmode,    // mode for printing
00065                                  bool     json      // JSON output
00066                                  );
00067
00068 // formatting of action flags for output
00069 std::string tr_formatActionFlags(uint16_t flags,    // 16bit flags
00070                                  uint64_t delay,    // used in case action was delayed
00071                                  uint32_t pmode,    // mode for printing
00072                                  bool     json      // JSON output
00073                                  );
00074
00075 namespace saftlib {
00079     int wait_for_signal(int timeout_ms = -1);
00080
00081     typedef saftbus::SignalGroup SignalGroup;
00082 }
00083
00084
00085
00086 #endif /* #ifndef COMMON_FUNCTIONS_H */
```

8.26 Condition.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef CONDITION_H
00023 #define CONDITION_H
00024
00025 #include "Owned.hpp"
00026
00027 #include <saftbus/service.hpp>
00028
00029 #include <cstdint>
00030 #include <string>
00031
00032 namespace saftlib {
00033
00034 class ActionSink;
00035
00042 class Condition  : public Owned
00043 {
00044   public:
00045     // if the created with active=true, you must manually run compile() on TimingReceiver
00046     Condition(ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t
      offset, uint32_t tag, saftbus::Container *container = nullptr);
```

```
00047     virtual ~Condition() = default;
00048
00049
00056     // @saftbus-export
00057     uint64_t getID() const;
00058     // @saftbus-export
00059     void setID(uint64_t val);
00060
00067     // @saftbus-export
00068     uint64_t getMask() const;
00069     // @saftbus-export
00070     void setMask(uint64_t val);
00071
00072
00076     // @saftbus-export
00077     int64_t getOffset() const;
00078     // @saftbus-export
00079     void setOffset(int64_t val);
00080
00084     // @saftbus-export
00085     bool getAcceptLate() const;
00086     // @saftbus-export
00087     void setAcceptLate(bool val);
00088
00092     // @saftbus-export
00093     bool getAcceptEarly() const;
00094     // @saftbus-export
00095     void setAcceptEarly(bool val);
00096
00100     // @saftbus-export
00101     bool getAcceptConflict() const;
00102     // @saftbus-export
00103     void setAcceptConflict(bool val);
00104
00108     // @saftbus-export
00109     bool getAcceptDelayed() const;
00110     // @saftbus-export
00111     void setAcceptDelayed(bool val);
00112
00121     // @saftbus-export
00122     bool getActive() const;
00123     // @saftbus-export
00124     void setActive(bool val);
00125
00126
00127     std::string &getObjectPath() { return objectPath; }
00128
00129
00130     // used by TimingReceiver and ActionSink
00131     uint32_t getRawTag() const { return tag; }
00132     void setRawActive(bool val) { active = val; }
00133
00134
00135     unsigned getNumber() const { return number; }
00136   protected:
00137
00138     std::string objectPath;
00139     ActionSink* sink;
00140     unsigned number;
00141
00142     uint64_t id;
00143     uint64_t mask;
00144     int64_t offset;
00145     uint32_t tag;
00146     bool acceptLate, acceptEarly, acceptConflict, acceptDelayed;
00147     bool active;
00148 };
00149
00150 }
00151
00152 #endif
```

8.27 Condition_Proxy.hpp

```
00001 #ifndef Condition_PROXY_HPP_
00002 #define Condition_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "Owned_Proxy.hpp"
00009
```

```
00010 namespace saftlib {
00011
00018     class Condition_Proxy : public  Owned_Proxy   {
00019         static std::vector<std::string> gen_interface_names();
00020     public:
00021         Condition_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00022         static std::shared_ptr<Condition_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00023         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00030     // @saftbus-export
00031         uint64_t getID();
00032     // @saftbus-export
00033         void setID(uint64_t val);
00040     // @saftbus-export
00041         uint64_t getMask();
00042     // @saftbus-export
00043         void setMask(uint64_t val);
00047     // @saftbus-export
00048         int64_t getOffset();
00049     // @saftbus-export
00050         void setOffset(int64_t val);
00054     // @saftbus-export
00055         bool getAcceptLate();
00056     // @saftbus-export
00057         void setAcceptLate(bool val);
00061     // @saftbus-export
00062         bool getAcceptEarly();
00063     // @saftbus-export
00064         void setAcceptEarly(bool val);
00068     // @saftbus-export
00069         bool getAcceptConflict();
00070     // @saftbus-export
00071         void setAcceptConflict(bool val);
00075     // @saftbus-export
00076         bool getAcceptDelayed();
00077     // @saftbus-export
00078         void setAcceptDelayed(bool val);
00087     // @saftbus-export
00088         bool getActive();
00089     // @saftbus-export
00090         void setActive(bool val);
00091     private:
00092         int interface_no;
00093
00094     };
00095
00096 }
00097
00098 #endif
```

8.28 Condition_Service.hpp

```
00001 #ifndef Condition_SERVICE_HPP_
00002 #define Condition_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class Condition;
00012     class Condition_Service : public saftbus::Service {
00013     public:
00014         Condition* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef Condition DriverType;
00017         Condition_Service(Condition* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         Condition_Service();
00019         ~Condition_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.29 eb-forward.hpp

```
00001 /*  Copyright (C) 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Michael Reese <m.reese@gsi.de>
00004  *
00005  ********************************************************************************
00006  *  This library is free software; you can redistribute it and/or
00007  *  modify it under the terms of the GNU Lesser General Public
00008  *  License as published by the Free Software Foundation; either
00009  *  version 3 of the License, or (at your option) any later version.
00010  *
00011  *  This library is distributed in the hope that it will be useful,
00012  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  *  Lesser General Public License for more details.
00015  *
00016  *  You should have received a copy of the GNU Lesser General Public
00017  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00018  ********************************************************************************
00019  */
00020
00021 #ifndef SAFTLIB_EB_FORWARD_H
00022 #define SAFTLIB_EB_FORWARD_H
00023
00024 #include <string>
00025 #include <vector>
00026 #include <map>
00027
00028 #include <saftbus/loop.hpp>
00029
00030 #include "SdbDevice.hpp"
00031
00032 namespace saftlib {
00033
00041     class EB_Forward : public SdbDevice {
00042     public:
00043         EB_Forward(const std::string& eb_name, etherbone::Device &device);
00044         ~EB_Forward();
00045
00046         bool accept_connection(int condition);
00047
00050         std::string eb_forward_path();
00051
00052     private:
00053         void write_all(int fd, char *ptr, int size);
00054         void read_all(int fd, char *ptr, int size);
00055         void open_pts();
00056         int     _eb_device_fd, _pts_fd;
00057         saftbus::SourceHandle io_source;
00058
00059         std::vector<std::pair<std::string, std::string> > visu;
00060
00061         std::vector<uint8_t> request;  // data from eb-tool
00062         std::vector<uint8_t> response; // data from device
00063
00064     };
00065
00066
00067 }
00068
00069 #endif
```

8.30 eb-source.hpp

```
00001 #ifndef TR_EB_SOURCE_HPP_
00002 #define TR_EB_SOURCE_HPP_
00003
00004 #ifndef ETHERBONE_THROWS
00005 #define ETHERBONE_THROWS 1
00006 #define __STDC_FORMAT_MACROS
00007 #define __STDC_CONSTANT_MACROS
00008 #endif
00009 #include <etherbone.h>
00010
00011 #include <memory>
00012 #include <vector>
00013
00014 #include <poll.h>
00015
00016 #include <saftbus/loop.hpp>
00017
00018 namespace saftlib {
```

```
00020     class EB_Source : public saftbus::Source
00021     {
00022     public:
00023         EB_Source(etherbone::Socket socket_);
00024 
00025         static int add_fd(eb_user_data_t, eb_descriptor_t, uint8_t mode);
00026         static int get_fd(eb_user_data_t, eb_descriptor_t, uint8_t mode);
00027 
00028         bool prepare(std::chrono::milliseconds &timeout_ms);
00029         bool check();
00030         bool dispatch();
00031         std::string type();
00032 
00033         ~EB_Source();
00034     private:
00035         etherbone::Socket socket;
00036         std::vector<pollfd> fds;
00037         std::vector<pollfd>::iterator fds_it;
00038         bool fds_it_valid;
00039     };
00040 
00041 }
00042 
00043 #endif
```

8.31 ECA.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021 
00022 #ifndef saftlib_ECA_DRIVER_HPP_
00023 #define saftlib_ECA_DRIVER_HPP_
00024 
00025 
00026 #ifndef ETHERBONE_THROWS
00027 #define ETHERBONE_THROWS 1
00028 #define __STDC_FORMAT_MACROS
00029 #define __STDC_CONSTANT_MACROS
00030 #endif
00031 #include <etherbone.h>
00032 
00033 #include <saftbus/service.hpp>
00034 
00035 #include "MsiDevice.hpp"
00036 
00037 #include <memory>
00038 
00039 
00040 namespace saftlib {
00041 
00042 class SAFTd;
00043 class IRQ;
00044 class OpenDevice;
00045 class ActionSink;
00046 class SoftwareActionSink;
00047 class Output;
00062 class ECA : public MsiDevice {
00063     friend class ActionSink;
00064 
00065     std::string        object_path;
00066     saftbus::Container *container;
00067     uint64_t sas_count; // counts number of SoftwareActionSinks
00068 
00069     unsigned channels;        // number of available ECA otput channels
00070     unsigned search_size;
```

```
00071     unsigned walker_size;
00072     unsigned max_conditions;
00073     unsigned used_conditions;
00074     std::vector<eb_address_t> channel_msis;
00075     std::vector<eb_address_t> queue_addresses;
00076     std::vector<uint16_t> most_full;
00077
00078
00079     std::vector<std::unique_ptr<IRQ> > channel_irqs;
00080
00081     std::vector<std::vector< std::unique_ptr<ActionSink> > > ECAchannels;
00082     std::vector< std::unique_ptr<ActionSink> >      *ECA_LINUX_channel; // a reference to the channels
      of type ECA_LINUX
00083     unsigned                                         ECA_LINUX_channel_index;
00084     unsigned                                         ECA_LINUX_channel_subchannels;
00085
00086     std::map<std::string, std::string > scubus_action_sinks; // a list of scubus_action_sinks that is
      created on construction and returned by getSCUbusActionSinks()
00087     std::map<std::string, std::string > ecpu_action_sinks; // a list of ecpu_action_sinks that is
      created on construction and returned by getEmbeddedCPUActionSinks()
00088     std::map<std::string, std::string > wbm_action_sinks;
00089
00090     void popMissingQueue(unsigned channel, unsigned num);
00091     void probeConfiguration();
00092     void prepareChannels();
00093     void msiHandler(eb_data_t msi, unsigned channel);
00094     void setHandler(unsigned channel, bool enable, eb_address_t address);
00095
00096     uint16_t getMostFull(int channel);
00097     eb_address_t get_base_address();
00098
00099 public:
00100     const std::string &get_object_path();
00101     etherbone::Device &get_device();
00102     void compile();
00103     // typedef std::pair<unsigned, unsigned> SinkKey; // (channel, num)
00104
00105
00106     ECA(SAFTd &saftd, etherbone::Device &device, const std::string &object_path, saftbus::Container
      *container);
00107     virtual ~ECA();
00108
00109
00111     bool addActionSink(int channel, std::unique_ptr<ActionSink> sink);
00112
00113     uint16_t updateMostFull(unsigned channel); // returns current fill
00114
00115     void removeSowftwareActionSink(SoftwareActionSink *sas);
00116
00123     // @saftbus-export
00124     uint64_t ReadRawCurrentTime() const;
00125
00137     // @saftbus-export
00138     std::string NewSoftwareActionSink(const std::string& name);
00139
00150     // @saftbus-export
00151     std::map< std::string, std::string > getSoftwareActionSinks() const;
00152
00156     // @saftbus-export
00157     std::map< std::string, std::string > getSCUbusActionSinks() const;
00158
00162     // @saftbus-export
00163     std::map< std::string, std::string > getEmbeddedCPUActionSinks() const;
00164
00168     // @saftbus-export
00169     std::map< std::string, std::string > getWbmActionSinks() const;
00170
00175     SoftwareActionSink *getSoftwareActionSink(const std::string & sas_obj_path);
00176
00178     // @saftbus-export
00179     void ToggleActive();
00180
00182     // @saftbus-export
00183     void InactivateAll();
00184
00190     // @saftbus-export
00191     std::map< std::string, std::string > getOutputs() const;
00192
00197     // @saftbus-export
00198     uint32_t getFree() const;
00199
00200     void resetMostFull(unsigned channel);
00201
00202
00203     Output *getOutput(const std::string &output_obj_path);
00204 };
00205
```

```
00206
00207 } // namespace
00208
00209 #endif
00210
```

8.32 eca_ac_wbm_regs.h

```
00001 #ifndef _ECA_AC_WBM_H_
00002 #define _ECA_AC_WBM_H_
00003
00004 //| Address Map ------------------------ slave ------------------------------------------|
00005 #define SLAVE_STATUS_GET         0x00  // r     _0x000000ff , Shows if the device is rdy/busy
00006 #define SLAVE_MAX_MACROS_GET     0x04  // r     _0xffffffff , Shows maximum number of macros
00007 #define SLAVE_MAX_SPACE_GET      0x08  // r     _0xffffffff , Shows maximum memory space
00008 #define SLAVE_ENABLE_GET         0x0c  // rw    _0x00000001 , Turns device on/off
00009 #define SLAVE_ENABLE_SET         0x10  // rw    _0x00000001 , ""
00010 #define SLAVE_ENABLE_CLR         0x14  // rw    _0x00000001 , ""
00011 #define SLAVE_EXEC_OWR           0x18  // wfs   _0x000000ff , Executes macro at idx
00012 #define SLAVE_LAST_EXEC_GET      0x1c  // r     _0x000000ff , Shows idx of last executed macro
00013 #define SLAVE_REC_OWR            0x20  // wfs   _0x000000ff , Records macro at idx
00014 #define SLAVE_LAST_REC_GET       0x24  // r     _0x000000ff , Shows idx of last recorded macro
00015 #define SLAVE_MACRO_QTY_GET      0x28  // r     _0x000000ff , Shows the number of macros in the ram
00016 #define SLAVE_SPACE_LEFT_GET     0x2c  // r     _0x0000ffff , Shows number of free spaces in the RAM
00017 #define SLAVE_CLEAR_ALL_OWR      0x30  // wsp   _0x00000001 , Clears all macros
00018 #define SLAVE_CLEAR_IDX_OWR      0x34  // wfs   _0x000000ff , Clears macro at idx
00019 #define SLAVE_REC_FIFO_OWR       0x38  // wf    _0xffffffff , Recording fifo. 3 word sequences: #ADR#
      #VAL# #META#
00020
00021 #endif
```

8.33 ECA_Event.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef saftlib_ECA_EVENT_DRIVER_HPP_
00023 #define saftlib_ECA_EVENT_DRIVER_HPP_
00024
00025
00026 #ifndef ETHERBONE_THROWS
00027 #define ETHERBONE_THROWS 1
00028 #define __STDC_FORMAT_MACROS
00029 #define __STDC_CONSTANT_MACROS
00030 #endif
00031 #include <etherbone.h>
00032
00033 #include <saftbus/service.hpp>
00034
00035 #include "SdbDevice.hpp"
00036
00037 #include <memory>
00038
00039 namespace saftlib {
00040
00041
00044 class ECA_Event : public SdbDevice {
00045
00046 public:
00047     ECA_Event(etherbone::Device &device, saftbus::Container *container = nullptr);
00048
```

```
00058     // @saftbus-export
00059     void InjectEventRaw(uint64_t event, uint64_t param, uint64_t time) const;
00060 };
00061
00062
00063 } // namespace
00064
00065 #endif
00066
```

8.34 ECA_Event_Proxy.hpp

```
00001 #ifndef ECA_Event_PROXY_HPP_
00002 #define ECA_Event_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008
00009 namespace saftlib {
00010
00013     class ECA_Event_Proxy : public virtual saftbus::Proxy
00014     {
00015         static std::vector<std::string> gen_interface_names();
00016     public:
00017         ECA_Event_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00018         static std::shared_ptr<ECA_Event_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00019         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00029     // @saftbus-export
00030         void InjectEventRaw(uint64_t event, uint64_t param, uint64_t time);
00031     private:
00032         int interface_no;
00033
00034     };
00035
00036 }
00037
00038 #endif
```

8.35 ECA_Event_Service.hpp

```
00001 #ifndef ECA_Event_SERVICE_HPP_
00002 #define ECA_Event_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class ECA_Event;
00012     class ECA_Event_Service : public saftbus::Service {
00013     public:
00014         ECA_Event* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef ECA_Event DriverType;
00017         ECA_Event_Service(ECA_Event* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         ECA_Event_Service();
00019         ~ECA_Event_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.36 eca_flags.h

```
00001
```

```
00020 #ifndef ECA_FLAGS_H
00021 #define ECA_FLAGS_H
00022
00023 #define ECA_LATE      0
00024 #define ECA_EARLY     1
00025 #define ECA_CONFLICT  2
00026 #define ECA_DELAYED   3
00027 #define ECA_VALID     4
00028 #define ECA_OVERFLOW  5
00029 #define ECA_MAX_FULL  6
00030
00031 #define ECA_LINUX         0
00032 #define ECA_WBM           1
00033 #define ECA_EMBEDDED_CPU  2
00034 #define ECA_SCUBUS        3
00035
00036 #endif
```

8.37 ECA_Proxy.hpp

```
00001 #ifndef ECA_PROXY_HPP_
00002 #define ECA_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008
00009 namespace saftlib {
00010
00025     class ECA_Proxy : public virtual saftbus::Proxy
00026     {
00027         static std::vector<std::string> gen_interface_names();
00028     public:
00029         ECA_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00030         static std::shared_ptr<ECA_Proxy> create(const std::string &object_path, saftbus::SignalGroup
      &signal_group = saftbus::SignalGroup::get_global());
00031         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00038     // @saftbus-export
00039         uint64_t ReadRawCurrentTime();
00051     // @saftbus-export
00052         std::string NewSoftwareActionSink(const std::string& name);
00063     // @saftbus-export
00064         std::map< std::string, std::string > getSoftwareActionSinks();
00068     // @saftbus-export
00069         std::map< std::string, std::string > getSCUbusActionSinks();
00073     // @saftbus-export
00074         std::map< std::string, std::string > getEmbeddedCPUActionSinks();
00078     // @saftbus-export
00079         std::map< std::string, std::string > getWbmActionSinks();
00081     // @saftbus-export
00082         void ToggleActive();
00084     // @saftbus-export
00085         void InactivateAll();
00091     // @saftbus-export
00092         std::map< std::string, std::string > getOutputs();
00097     // @saftbus-export
00098         uint32_t getFree();
00099     private:
00100         int interface_no;
00101
00102     };
00103
00104 }
00105
00106 #endif
```

8.38 src/eca_queue_regs.h File Reference

Register map for Wishbone interface of VHDL entity <eca_queue_auto>

Macros

- #define **ECA_QUEUE_SDB_VENDOR_ID** 0x00000651
- #define **ECA_QUEUE_SDB_DEVICE_ID** 0xd5a3faea
- #define **ECA_QUEUE_QUEUE_ID_GET** 0x00
- #define **ECA_QUEUE_POP_OWR** 0x04
- #define **ECA_QUEUE_FLAGS_GET** 0x08
- #define **ECA_QUEUE_NUM_GET** 0x0c
- #define **ECA_QUEUE_EVENT_ID_HI_GET** 0x10
- #define **ECA_QUEUE_EVENT_ID_LO_GET** 0x14
- #define **ECA_QUEUE_PARAM_HI_GET** 0x18
- #define **ECA_QUEUE_PARAM_LO_GET** 0x1c
- #define **ECA_QUEUE_TAG_GET** 0x20
- #define **ECA_QUEUE_TEF_GET** 0x24
- #define **ECA_QUEUE_DEADLINE_HI_GET** 0x28
- #define **ECA_QUEUE_DEADLINE_LO_GET** 0x2c
- #define **ECA_QUEUE_EXECUTED_HI_GET** 0x30
- #define **ECA_QUEUE_EXECUTED_LO_GET** 0x34

8.38.1 Detailed Description

Register map for Wishbone interface of VHDL entity <eca_queue_auto>

Author

Wesley W. Terpstra w.terpstra@gsi.de

DesignUnit eca_queue

Author

Wesley W. Terpstra w.terpstra@gsi.de

Date

08/04/2016

Version

2.0

Copyright

8.39 eca_queue_regs.h

Go to the documentation of this file.

```
00001
00030 #ifndef _ECA_QUEUE_H_
00031 #define _ECA_QUEUE_H_
00032
00033   #define ECA_QUEUE_SDB_VENDOR_ID 0x00000651
00034   #define ECA_QUEUE_SDB_DEVICE_ID 0xd5a3faea
00035
00036   #define ECA_QUEUE_QUEUE_ID_GET     0x00  //ro,  8 b, The index of a_channel_o from the ECA to which
      this queue is connected (set channel_select=queue_id+1)
00037   #define ECA_QUEUE_POP_OWR          0x04  //wo,  1 b, Pop action from the channel's queue
00038   #define ECA_QUEUE_FLAGS_GET        0x08  //ro,  5 b, Error flags for this action(0=late, 1=early,
      2=conflict, 3=delayed, 4=valid)
00039   #define ECA_QUEUE_NUM_GET          0x0c  //ro,  8 b, Subchannel target
00040   #define ECA_QUEUE_EVENT_ID_HI_GET  0x10  //ro, 32 b, Event ID (high word)
00041   #define ECA_QUEUE_EVENT_ID_LO_GET  0x14  //ro, 32 b, Event ID (low word)
00042   #define ECA_QUEUE_PARAM_HI_GET     0x18  //ro, 32 b, Parameter (high word)
00043   #define ECA_QUEUE_PARAM_LO_GET     0x1c  //ro, 32 b, Parameter (low word)
00044   #define ECA_QUEUE_TAG_GET          0x20  //ro, 32 b, Tag from the condition
00045   #define ECA_QUEUE_TEF_GET          0x24  //ro, 32 b, Timing extension field
00046   #define ECA_QUEUE_DEADLINE_HI_GET  0x28  //ro, 32 b, Deadline (high word)
00047   #define ECA_QUEUE_DEADLINE_LO_GET  0x2c  //ro, 32 b, Deadline (low word)
00048   #define ECA_QUEUE_EXECUTED_HI_GET  0x30  //ro, 32 b, Actual execution time (high word)
00049   #define ECA_QUEUE_EXECUTED_LO_GET  0x34  //ro, 32 b, Actual execution time (low word)
00050
00051 #endif
```

8.40 src/eca_regs.h File Reference

Register map for Wishbone interface of VHDL entity <eca_auto>

Macros

- #define **ECA_SDB_VENDOR_ID** 0x00000651
- #define **ECA_SDB_DEVICE_ID** 0xb2afc251
- #define **ECA_CHANNELS_GET** 0x00
- #define **ECA_SEARCH_CAPACITY_GET** 0x04
- #define **ECA_WALKER_CAPACITY_GET** 0x08
- #define **ECA_LATENCY_GET** 0x0c
- #define **ECA_OFFSET_BITS_GET** 0x10
- #define **ECA_FLIP_ACTIVE_OWR** 0x14
- #define **ECA_TIME_HI_GET** 0x18
- #define **ECA_TIME_LO_GET** 0x1c
- #define **ECA_SEARCH_SELECT_RW** 0x20
- #define **ECA_SEARCH_RW_FIRST_RW** 0x24
- #define **ECA_SEARCH_RW_EVENT_HI_RW** 0x28
- #define **ECA_SEARCH_RW_EVENT_LO_RW** 0x2c
- #define **ECA_SEARCH_WRITE_OWR** 0x30
- #define **ECA_SEARCH_RO_FIRST_GET** 0x34
- #define **ECA_SEARCH_RO_EVENT_HI_GET** 0x38
- #define **ECA_SEARCH_RO_EVENT_LO_GET** 0x3c
- #define **ECA_WALKER_SELECT_RW** 0x40
- #define **ECA_WALKER_RW_NEXT_RW** 0x44
- #define **ECA_WALKER_RW_OFFSET_HI_RW** 0x48
- #define **ECA_WALKER_RW_OFFSET_LO_RW** 0x4c
- #define **ECA_WALKER_RW_TAG_RW** 0x50
- #define **ECA_WALKER_RW_FLAGS_RW** 0x54
- #define **ECA_WALKER_RW_CHANNEL_RW** 0x58

- #define **ECA_WALKER_RW_NUM_RW** 0x5c
- #define **ECA_WALKER_WRITE_OWR** 0x60
- #define **ECA_WALKER_RO_NEXT_GET** 0x64
- #define **ECA_WALKER_RO_OFFSET_HI_GET** 0x68
- #define **ECA_WALKER_RO_OFFSET_LO_GET** 0x6c
- #define **ECA_WALKER_RO_TAG_GET** 0x70
- #define **ECA_WALKER_RO_FLAGS_GET** 0x74
- #define **ECA_WALKER_RO_CHANNEL_GET** 0x78
- #define **ECA_WALKER_RO_NUM_GET** 0x7c
- #define **ECA_CHANNEL_SELECT_RW** 0x80
- #define **ECA_CHANNEL_NUM_SELECT_RW** 0x84
- #define **ECA_CHANNEL_CODE_SELECT_RW** 0x88
- #define **ECA_CHANNEL_TYPE_GET** 0x90
- #define **ECA_CHANNEL_MAX_NUM_GET** 0x94
- #define **ECA_CHANNEL_CAPACITY_GET** 0x98
- #define **ECA_CHANNEL_MSI_SET_ENABLE_OWR** 0x9c
- #define **ECA_CHANNEL_MSI_GET_ENABLE_GET** 0xa0
- #define **ECA_CHANNEL_MSI_SET_TARGET_OWR** 0xa4
- #define **ECA_CHANNEL_MSI_GET_TARGET_GET** 0xa8
- #define **ECA_CHANNEL_MOSTFULL_ACK_GET** 0xac
- #define **ECA_CHANNEL_MOSTFULL_CLEAR_GET** 0xb0
- #define **ECA_CHANNEL_VALID_COUNT_GET** 0xb4
- #define **ECA_CHANNEL_OVERFLOW_COUNT_GET** 0xb8
- #define **ECA_CHANNEL_FAILED_COUNT_GET** 0xbc
- #define **ECA_CHANNEL_EVENT_ID_HI_GET** 0xc0
- #define **ECA_CHANNEL_EVENT_ID_LO_GET** 0xc4
- #define **ECA_CHANNEL_PARAM_HI_GET** 0xc8
- #define **ECA_CHANNEL_PARAM_LO_GET** 0xcc
- #define **ECA_CHANNEL_TAG_GET** 0xd0
- #define **ECA_CHANNEL_TEF_GET** 0xd4
- #define **ECA_CHANNEL_DEADLINE_HI_GET** 0xd8
- #define **ECA_CHANNEL_DEADLINE_LO_GET** 0xdc
- #define **ECA_CHANNEL_EXECUTED_HI_GET** 0xe0
- #define **ECA_CHANNEL_EXECUTED_LO_GET** 0xe4

8.40.1 Detailed Description

Register map for Wishbone interface of VHDL entity <eca_auto>

Author

Wesley W. Terpstra w.terpstra@gsi.de

DesignUnit eca

Author

Wesley W. Terpstra w.terpstra@gsi.de

Date

08/04/2016

Version

2.0

Copyright

8.41 eca_regs.h

Go to the documentation of this file.

```
00001
00030 #ifndef _ECA_H_
00031 #define _ECA_H_
00032
00033   #define ECA_SDB_VENDOR_ID 0x00000651
00034   #define ECA_SDB_DEVICE_ID 0xb2afc251
00035
00036   #define ECA_CHANNELS_GET              0x00  //ro,  8 b, Number of channels implemented by the ECA,
      including the internal IO channel #0
00037   #define ECA_SEARCH_CAPACITY_GET       0x04  //ro, 16 b, Total number of search table entries per
      active page
00038   #define ECA_WALKER_CAPACITY_GET       0x08  //ro, 16 b, Total number of walker table entries per
      active page
00039   #define ECA_LATENCY_GET               0x0c  //ro, 32 b, Delay in ticks (typically nanoseconds)
      between an event's arrival at the ECA and its earliest possible execution as an action
00040   #define ECA_OFFSET_BITS_GET           0x10  //ro,  8 b, Actions scheduled for execution with a
      delay in ticks exceeding offset_bits are executed early
00041   #define ECA_FLIP_ACTIVE_OWR           0x14  //wo,  1 b, Flip the active search and walker tables
      with the inactive tables
00042   #define ECA_TIME_HI_GET               0x18  //ro, 32 b, Ticks (nanoseconds) since Jan 1, 1970 (high
      word)
00043   #define ECA_TIME_LO_GET               0x1c  //ro, 32 b, Ticks (nanoseconds) since Jan 1, 1970 (low
      word)
00044   #define ECA_SEARCH_SELECT_RW          0x20  //rw, 16 b, Read/write this record in the inactive
      search table
00045   #define ECA_SEARCH_RW_FIRST_RW        0x24  //rw, 16 b, Scratch register to be written to
      search_ro_first
00046   #define ECA_SEARCH_RW_EVENT_HI_RW     0x28  //rw, 32 b, Scratch register to be written to
      search_ro_event_hi
00047   #define ECA_SEARCH_RW_EVENT_LO_RW     0x2c  //rw, 32 b, Scratch register to be written to
      search_ro_event_lo
00048   #define ECA_SEARCH_WRITE_OWR          0x30  //wo,  1 b, Store the scratch registers to the inactive
      search table record search_select
00049   #define ECA_SEARCH_RO_FIRST_GET       0x34  //ro, 16 b, The first walker entry to execute if an
      event matches this record in the search table
00050   #define ECA_SEARCH_RO_EVENT_HI_GET    0x38  //ro, 32 b, Event IDs greater than or equal to this
      value match this search table record (high word)
00051   #define ECA_SEARCH_RO_EVENT_LO_GET    0x3c  //ro, 32 b, Event IDs greater than or equal to this
      value match this search table record (low word)
00052   #define ECA_WALKER_SELECT_RW          0x40  //rw, 16 b, Read/write this record in the inactive
      walker table
00053   #define ECA_WALKER_RW_NEXT_RW         0x44  //rw, 16 b, Scratch register to be written to
      walker_ro_next
00054   #define ECA_WALKER_RW_OFFSET_HI_RW    0x48  //rw, 32 b, Scratch register to be written to
      walker_ro_offset_hi
00055   #define ECA_WALKER_RW_OFFSET_LO_RW    0x4c  //rw, 32 b, Scratch register to be written to
      walker_ro_offset_lo
00056   #define ECA_WALKER_RW_TAG_RW          0x50  //rw, 32 b, Scratch register to be written to
      walker_ro_tag
00057   #define ECA_WALKER_RW_FLAGS_RW        0x54  //rw,  4 b, Scratch register to be written to
      walker_ro_flags
00058   #define ECA_WALKER_RW_CHANNEL_RW      0x58  //rw,  8 b, Scratch register to be written to
      walker_ro_channel
```

```
00059   #define ECA_WALKER_RW_NUM_RW           0x5c  //rw,  8 b, Scratch register to be written to
      walker_ro_num
00060   #define ECA_WALKER_WRITE_OWR           0x60  //wo,  1 b, Store the scratch registers to the inactive
      walker table record walker_select
00061   #define ECA_WALKER_RO_NEXT_GET         0x64  //ro, 16 b, The next walker entry to execute after this
      record (0xffff = end of list)
00062   #define ECA_WALKER_RO_OFFSET_HI_GET    0x68  //ro, 32 b, The resulting action's deadline is the
      event timestamp plus this offset (high word)
00063   #define ECA_WALKER_RO_OFFSET_LO_GET    0x6c  //ro, 32 b, The resulting action's deadline is the
      event timestamp plus this offset (low word)
00064   #define ECA_WALKER_RO_TAG_GET          0x70  //ro, 32 b, The resulting actions's tag
00065   #define ECA_WALKER_RO_FLAGS_GET        0x74  //ro,  4 b, Execute the resulting action even if it
      suffers from the errors set in this flag register
00066   #define ECA_WALKER_RO_CHANNEL_GET      0x78  //ro,  8 b, The channel to which the resulting action
      will be sent
00067   #define ECA_WALKER_RO_NUM_GET          0x7c  //ro,  8 b, The subchannel to which the resulting
      action will be sent
00068   #define ECA_CHANNEL_SELECT_RW          0x80  //rw,  8 b, Read/clear this channel
00069   #define ECA_CHANNEL_NUM_SELECT_RW      0x84  //rw,  8 b, Read/clear this subchannel
00070   #define ECA_CHANNEL_CODE_SELECT_RW     0x88  //rw,  2 b, Read/clear this error condition (0=late,
      1=early, 2=conflict, 3=delayed)
00071   #define ECA_CHANNEL_TYPE_GET           0x90  //ro, 32 b, Type of the selected channel (0=io,
      1=linux, 2=wbm, ...)
00072   #define ECA_CHANNEL_MAX_NUM_GET        0x94  //ro,  8 b, Total number of subchannels supported by
      the selected channel
00073   #define ECA_CHANNEL_CAPACITY_GET       0x98  //ro, 16 b, Total number of actions which may be
      enqueued by the selected channel at a time
00074   #define ECA_CHANNEL_MSI_SET_ENABLE_OWR 0x9c  //wo,  1 b, Turn on/off MSI messages for the selected
      channel
00075   #define ECA_CHANNEL_MSI_GET_ENABLE_GET 0xa0  //ro,  1 b, Check if MSI messages are enabled for the
      selected channel
00076   #define ECA_CHANNEL_MSI_SET_TARGET_OWR 0xa4  //wo, 32 b, Set the destination MSI address for the
      selected channel (only possible while it has MSIs disabled)
00077   #define ECA_CHANNEL_MSI_GET_TARGET_GET 0xa8  //ro, 32 b, Get the destination MSI address for the
      selected channel
00078   #define ECA_CHANNEL_MOSTFULL_ACK_GET   0xac  //ro, 32 b, Read the selected channel's fill status
      (used_now«16 | used_most), MSI=(6«16) will be sent if used_most changes
00079   #define ECA_CHANNEL_MOSTFULL_CLEAR_GET 0xb0  //ro, 32 b, Read and clear the selected channel's fill
      status (used_now«16 | used_most), MSI=(6«16) will be sent if used_most changes
00080   #define ECA_CHANNEL_VALID_COUNT_GET    0xb4  //ro, 32 b, Read and clear the number of actions output
      by the selected subchannel, MSI=(4«16|num) will be sent when the count becomes non-zero
00081   #define ECA_CHANNEL_OVERFLOW_COUNT_GET 0xb8  //ro, 32 b, Read and clear the number of actions which
      could not be enqueued to the selected full channel which were destined for the selected subchannel,
      MSI=(5«16|num) will be sent when the count becomes non-zero
00082   #define ECA_CHANNEL_FAILED_COUNT_GET   0xbc  //ro, 32 b, Read and clear the number of actions with
      the selected error code which were destined for the selected subchannel, MSI=(code«16|num) will be
      sent when the count becomes non-zero
00083   #define ECA_CHANNEL_EVENT_ID_HI_GET    0xc0  //ro, 32 b, The event ID of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read (high word)
00084   #define ECA_CHANNEL_EVENT_ID_LO_GET    0xc4  //ro, 32 b, The event ID of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read (low word)
00085   #define ECA_CHANNEL_PARAM_HI_GET       0xc8  //ro, 32 b, The parameter of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read (high word)
00086   #define ECA_CHANNEL_PARAM_LO_GET       0xcc  //ro, 32 b, The parameter of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read (low word)
00087   #define ECA_CHANNEL_TAG_GET            0xd0  //ro, 32 b, The tag of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read
00088   #define ECA_CHANNEL_TEF_GET            0xd4  //ro, 32 b, The TEF of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read
00089   #define ECA_CHANNEL_DEADLINE_HI_GET    0xd8  //ro, 32 b, The deadline of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read (high word)
00090   #define ECA_CHANNEL_DEADLINE_LO_GET    0xdc  //ro, 32 b, The deadline of the first action with the
      selected error code on the selected subchannel, cleared when channel_failed_count is read (low word)
00091   #define ECA_CHANNEL_EXECUTED_HI_GET    0xe0  //ro, 32 b, The actual execution time of the first
      action with the selected error code on the selected subchannel, cleared when channel_failed_count is
      read (high word)
00092   #define ECA_CHANNEL_EXECUTED_LO_GET    0xe4  //ro, 32 b, The actual execution time of the first
      action with the selected error code on the selected subchannel, cleared when channel_failed_count is
      read (low word)
00093
00094 #endif
```

8.42 ECA_Service.hpp

```
00001 #ifndef ECA_SERVICE_HPP_
00002 #define ECA_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
```

```
00010
00011     class ECA;
00012     class ECA_Service : public saftbus::Service {
00013     public:
00014         ECA* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef ECA DriverType;
00017         ECA_Service(ECA* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         ECA_Service();
00019         ~ECA_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.43 ECA_TLU.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_ECA_TLU_HPP_
00023 #define saftlib_ECA_TLU_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include <memory>
00033 #include <map>
00034
00035 #include <saftbus/service.hpp>
00036
00037 #include "SdbDevice.hpp"
00038
00039 namespace saftlib {
00040
00041 class Input;
00042
00044 class ECA_TLU : public SdbDevice {
00045     saftbus::Container *container;
00046     std::vector<std::unique_ptr<Input> > inputs;
00047
00048 public:
00049     ECA_TLU(etherbone::Device &device, saftbus::Container *container = nullptr);
00050     ~ECA_TLU();
00051
00053     void addInput(std::unique_ptr<Input> input);
00054
00055     // @saftbus-export
00056     std::map< std::string, std::string > getInputs() const;
00057
00058     void configInput(unsigned channel,
00059                      bool enable,
00060                      uint64_t event,
00061                      uint32_t stable);
00062 };
00063
```

```
00064 }
00065
00066 #endif
```

8.44 ECA_TLU_Proxy.hpp

```
00001 #ifndef ECA_TLU_PROXY_HPP_
00002 #define ECA_TLU_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008
00009 namespace saftlib {
00010
00012     class ECA_TLU_Proxy : public virtual saftbus::Proxy
00013     {
00014         static std::vector<std::string> gen_interface_names();
00015     public:
00016         ECA_TLU_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00017         static std::shared_ptr<ECA_TLU_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00018         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00019     // @saftbus-export
00020         std::map< std::string, std::string > getInputs();
00021     private:
00022         int interface_no;
00023
00024     };
00025
00026 }
00027
00028 #endif
```

8.45 src/eca_tlu_regs.h File Reference

Register map for Wishbone interface of VHDL entity <eca_tlu_auto>

Macros

- #define **ECA_TLU_SDB_VENDOR_ID** 0x00000651
- #define **ECA_TLU_SDB_DEVICE_ID** 0x7c82afbc
- #define **ECA_TLU_NUM_INPUTS_GET** 0x00
- #define **ECA_TLU_INPUT_SELECT_RW** 0x04
- #define **ECA_TLU_ENABLE_RW** 0x08
- #define **ECA_TLU_STABLE_RW** 0x0c
- #define **ECA_TLU_EVENT_HI_RW** 0x10
- #define **ECA_TLU_EVENT_LO_RW** 0x14
- #define **ECA_TLU_WRITE_OWR** 0x18

8.45.1 Detailed Description

Register map for Wishbone interface of VHDL entity <eca_tlu_auto>

DesignUnit eca_tlu

Author

Wesley W. Terpstra w.terpstra@gsi.de

Date

15/04/2016

Version

2.0

Copyright

8.46 eca_tlu_regs.h

Go to the documentation of this file.

```
00001
00011 #ifndef _ECA_TLU_H_
00012 #define _ECA_TLU_H_
00013
00014   #define ECA_TLU_SDB_VENDOR_ID 0x00000651
00015   #define ECA_TLU_SDB_DEVICE_ID 0x7c82afbc
00016
00017   #define ECA_TLU_NUM_INPUTS_GET   0x00  //ro,  8 b, Total number of inputs attached to the TLU
00018   #define ECA_TLU_INPUT_SELECT_RW  0x04  //rw,  8 b, Write the configuration of this input
00019   #define ECA_TLU_ENABLE_RW        0x08  //rw,  1 b, Will this input generate timing events on an edge
00020   #define ECA_TLU_STABLE_RW        0x0c  //rw, 32 b, Signal must be high/low for stable cycles to be
      counted as a valid transition
00021   #define ECA_TLU_EVENT_HI_RW      0x10  //rw, 32 b, Timing Event to generate (high word)
00022   #define ECA_TLU_EVENT_LO_RW      0x14  //rw, 32 b, Timing Event to generate (low word), lowest bit
      is replaced with the edge of the transition
00023   #define ECA_TLU_WRITE_OWR        0x18  //wo,  1 b, Write register contents to TLU configuration
00024
00025 #endif
```

8.47 ECA_TLU_Service.hpp

```
00001 #ifndef ECA_TLU_SERVICE_HPP_
00002 #define ECA_TLU_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class ECA_TLU;
00012     class ECA_TLU_Service : public saftbus::Service {
00013     public:
00014         ECA_TLU* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef ECA_TLU DriverType;
00017         ECA_TLU_Service(ECA_TLU* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         ECA_TLU_Service();
00019         ~ECA_TLU_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.48 EmbeddedCPUActionSink.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef SAFTLIB_EMBEDDEDCPU_ACTION_SINK_HPP_
00023 #define SAFTLIB_EMBEDDEDCPU_ACTION_SINK_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "ActionSink.hpp"
00033
00034 namespace saftlib {
00035
00045 class EmbeddedCPUActionSink : public ActionSink {
00046     public:
00047         EmbeddedCPUActionSink(ECA &eca
00048                         , const std::string &object_path
00049                         , const std::string &name
00050                         , unsigned channel
00051                         , saftbus::Container *container = nullptr);
00052
00068         // @saftbus-export
00069         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t
      tag);
00070
00071
00072 };
00073
00074 }
00075
00076 #endif
```

8.49 EmbeddedCPUActionSink_Proxy.hpp

```
00001 #ifndef EmbeddedCPUActionSink_PROXY_HPP_
00002 #define EmbeddedCPUActionSink_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "ActionSink_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00021     class EmbeddedCPUActionSink_Proxy : public  ActionSink_Proxy     {
00022         static std::vector<std::string> gen_interface_names();
00023     public:
00024         EmbeddedCPUActionSink_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group
      = saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00025         static std::shared_ptr<EmbeddedCPUActionSink_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00026         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00042         // @saftbus-export
00043         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t
      tag);
00044     private:
00045         int interface_no;
```

```
00046
00047     };
00048
00049 }
00050
00051 #endif
```

8.50 EmbeddedCPUActionSink_Service.hpp

```
00001 #ifndef EmbeddedCPUActionSink_SERVICE_HPP_
00002 #define EmbeddedCPUActionSink_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class EmbeddedCPUActionSink;
00012     class EmbeddedCPUActionSink_Service : public saftbus::Service {
00013     public:
00014         EmbeddedCPUActionSink* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef EmbeddedCPUActionSink DriverType;
00017         EmbeddedCPUActionSink_Service(EmbeddedCPUActionSink* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         EmbeddedCPUActionSink_Service();
00019         ~EmbeddedCPUActionSink_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void OverflowCount_dispatch_function(uint64_t arg_1);
00023         sigc::connection OverflowCount_connection;
00024         void ActionCount_dispatch_function(uint64_t arg_1);
00025         sigc::connection ActionCount_connection;
00026         void LateCount_dispatch_function(uint64_t arg_1);
00027         sigc::connection LateCount_connection;
00028         void SigLate_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00029         sigc::connection SigLate_connection;
00030         void EarlyCount_dispatch_function(uint64_t arg_1);
00031         sigc::connection EarlyCount_connection;
00032         void SigEarly_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00033         sigc::connection SigEarly_connection;
00034         void ConflictCount_dispatch_function(uint64_t arg_1);
00035         sigc::connection ConflictCount_connection;
00036         void SigConflict_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00037         sigc::connection SigConflict_connection;
00038         void DelayedCount_dispatch_function(uint64_t arg_1);
00039         sigc::connection DelayedCount_connection;
00040         void SigDelayed_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00041         sigc::connection SigDelayed_connection;
00042         void Destroyed_dispatch_function();
00043         sigc::connection Destroyed_connection;
00044     };
00045
00046 }
00047
00048 #endif
```

8.51 EmbeddedCPUCondition.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef SAFTLIB_EMBEDDEDCPU_CONDITION_HPP_
00023 #define SAFTLIB_EMBEDDEDCPU_CONDITION_HPP_
00024
00025 #include <saftbus/service.hpp>
00026
00027 #include "Owned.hpp"
00028 #include "Condition.hpp"
00029 #include "ActionSink.hpp"
00030
00031 namespace saftlib {
00032
00033
00034 class EmbeddedCPUCondition_Service;
00035
00036
00043 class EmbeddedCPUCondition : public Condition {
00044 public:
00045     EmbeddedCPUCondition(ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask,
      int64_t offset, uint32_t tag, saftbus::Container *container = nullptr);
00046
00049     // @saftbus-export
00050     uint32_t getTag() const;
00053     // @saftbus-export
00054     void setTag(uint32_t val);
00055
00056     typedef EmbeddedCPUCondition_Service ServiceType;
00057 };
00058
00059 }
00060
00061 #endif
```

8.52 EmbeddedCPUCondition_Proxy.hpp

```
00001 #ifndef EmbeddedCPUCondition_PROXY_HPP_
00002 #define EmbeddedCPUCondition_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "Condition_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00018     class EmbeddedCPUCondition_Proxy : public  Condition_Proxy     {
00019         static std::vector<std::string> gen_interface_names();
00020     public:
00021         EmbeddedCPUCondition_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group
      = saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00022         static std::shared_ptr<EmbeddedCPUCondition_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00023         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00026     // @saftbus-export
00027         uint32_t getTag();
00030     // @saftbus-export
00031         void setTag(uint32_t val);
00032     private:
00033         int interface_no;
00034
00035     };
00036
00037 }
00038
00039 #endif
```

8.53 EmbeddedCPUCondition_Service.hpp

```
00001 #ifndef EmbeddedCPUCondition_SERVICE_HPP_
00002 #define EmbeddedCPUCondition_SERVICE_HPP_
```

```
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class EmbeddedCPUCondition;
00012     class EmbeddedCPUCondition_Service : public saftbus::Service {
00013     public:
00014         EmbeddedCPUCondition* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef EmbeddedCPUCondition DriverType;
00017         EmbeddedCPUCondition_Service(EmbeddedCPUCondition* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         EmbeddedCPUCondition_Service();
00019         ~EmbeddedCPUCondition_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.54 EventSource.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef saftlib_EVENTSOURCE_HPP_
00023 #define saftlib_EVENTSOURCE_HPP_
00024
00025 #include "Owned.hpp"
00026
00027 namespace saftlib {
00028
00029 class TimingReceiver;
00030
00033 class EventSource : public Owned {
00034     std::string object_path;
00035     std::string object_name;
00036 public:
00037     EventSource(const std::string &object_path, const std::string &name, saftbus::Container
      *container);
00038
00042     // @saftbus-export
00043     virtual uint64_t getResolution() const = 0;
00044
00048     // @saftbus-export
00049     virtual uint32_t getEventBits() const = 0;
00050
00051
00055     // @saftbus-export
00056     virtual bool getEventEnable() const = 0;
00057     // @saftbus-export
00058     virtual void setEventEnable(bool val) = 0;
00059
00063     // @saftbus-export
00064     virtual uint64_t getEventPrefix() const = 0;
00065     // @saftbus-export
```

```
00066     virtual void setEventPrefix(uint64_t val) = 0;
00067
00068     std::string getObjectPath() const;
00069     std::string getObjectName() const;
00070 };
00071
00072 }
00073
00074 #endif
```

8.55 EventSource_Proxy.hpp

```
00001 #ifndef EventSource_PROXY_HPP_
00002 #define EventSource_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "Owned_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00014     class EventSource_Proxy : public  Owned_Proxy   {
00015         static std::vector<std::string> gen_interface_names();
00016     public:
00017         EventSource_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00018         static std::shared_ptr<EventSource_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00019         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00023     // @saftbus-export
00024         virtual uint64_t getResolution();
00028     // @saftbus-export
00029         virtual uint32_t getEventBits();
00033     // @saftbus-export
00034         virtual bool getEventEnable();
00035     // @saftbus-export
00036         virtual void setEventEnable(bool val);
00040     // @saftbus-export
00041         virtual uint64_t getEventPrefix();
00042     // @saftbus-export
00043         virtual void setEventPrefix(uint64_t val);
00044     private:
00045         int interface_no;
00046
00047     };
00048
00049 }
00050
00051 #endif
```

8.56 EventSource_Service.hpp

```
00001 #ifndef EventSource_SERVICE_HPP_
00002 #define EventSource_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class EventSource;
00012     class EventSource_Service : public saftbus::Service {
00013     public:
00014         EventSource* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef EventSource DriverType;
00017         EventSource_Service(EventSource* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         EventSource_Service();
00019         ~EventSource_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Destroyed_dispatch_function();
```

```
00023         sigc::connection Destroyed_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.57 fg_regs.h

```
00001
00020 #ifndef FG_REGS_H
00021 #define FG_REGS_H
00022
00023 // Locate shared memory of processors
00024 #define LM32_RAM_USER_VENDOR      0x651       //vendor ID
00025 #define LM32_RAM_USER_PRODUCT     0x54111351  //product ID
00026 #define LM32_RAM_USER_VMAJOR      1           //major revision
00027 #define LM32_RAM_USER_VMINOR      1           //minor revision
00028
00029 #define LM32_CLUSTER_ROM_VENDOR   0x651
00030 #define LM32_CLUSTER_ROM_PRODUCT  0x10040086
00031
00032 #define MSI_MAILBOX_VENDOR        0x651
00033 #define MSI_MAILBOX_PRODUCT       0xfab0bdd8
00034
00035 // Shared memory base address
00036 #define SHM_BASE          0x500
00037 #define BOARD_ID          0x0   // 64-bit
00038 #define EXT_ID            0x8   // 64-bit
00039 #define BACKPLANE_ID      0x10  // 64-bit
00040 #define BOARD_TEMP        0x18  // 32-bit ...
00041 #define EXT_TEMP          0x1c
00042 #define BACKPLACE_TEMP    0x20
00043 #define FG_MAGIC_NUMBER   0x24  // 0xdeadbeef
00044 #define FG_VERSION        0x28  // expect version 3 or 4
00045 #define FG_MB_SLOT        0x2c  // mb slot for host=>lm32
00046 #define FG_NUM_CHANNELS   0x30
00047 #define FG_BUFFER_SIZE    0x34
00048 #define FG_MACROS         0x38  // 256 entries of (hi..lo): slot, device, version, output-bits
00049 #define FG_BUSY           0x60b8 // firmware writes 0 here if scan of channels is done (fw-version 4)
00050 #define FG_SCAN_DONE      0x100c0 // firmware writes 0 here if scan of channels is done (fw-version 3)
00051
00052 #define FG_MACROS_SIZE  256
00053
00054 #define FG_REGS_BASE_   0x438   // FG_MACROS + 4 * 256 + 4
00055 #define FG_WPTR         0x0
00056 #define FG_RPTR         0x4
00057 #define FG_MBX_SLOT     0x8
00058 #define FG_MACRO_NUM    0xc
00059 #define FG_RAMP_COUNT   0x10
00060 #define FG_TAG          0x14
00061 #define FG_STATE        0x18
00062 #define FG_REGS_SIZE    0x1c
00063
00064 // channel, num_channels
00065 #define FG_REGS_BASE(c,n) (FG_REGS_BASE_+FG_REGS_SIZE*c)
00066
00067 #define PARAM_COEFF_AB  0x0 // high = coeff_a
00068 #define PARAM_COEFF_C   0x4
00069 #define PARAM_CONTROL   0x8   // 2..0=step, 5..3=freq, 11..6=shift_b, 17..12=shifta
00070 #define PARAM_SIZE      0xc
00071
00072 // channel, index, num_channels, buffer_size
00073 #define FG_BUFF_BASE(c,i,n,s) (FG_REGS_BASE(n,n)+(i+c*s)*PARAM_SIZE)
00074
00075
00076 // Software interrupt numbers; host=>lm32
00077 #define SWI_INIT_BUFFERS 0x00000000UL
00078 #define SWI_ENABLE       0x00020000UL
00079 #define SWI_DISABLE      0x00030000UL
00080 #define SWI_SCAN         0x00040000UL // cause firmware to scan all busses for fg channels
00081
00082 // interrupt payload; lm32=>host
00083 #define IRQ_DAT_REFILL          0
00084 #define IRQ_DAT_START           1
00085 #define IRQ_DAT_STOP_EMPTY      2
00086 #define IRQ_DAT_STOP_NOT_EMPTY  3
00087 #define IRQ_DAT_ARMED           4
00088 #define IRQ_DAT_DISARMED        5
00089
00090
00091 #endif
```

8.58 FunctionGenerator.hpp

```
00001 /*  Copyright (C) 2011-2016 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *
00005  ********************************************************************************
00006  *  This library is free software; you can redistribute it and/or
00007  *  modify it under the terms of the GNU Lesser General Public
00008  *  License as published by the Free Software Foundation; either
00009  *  version 3 of the License, or (at your option) any later version.
00010  *
00011  *  This library is distributed in the hope that it will be useful,
00012  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  *  Lesser General Public License for more details.
00015  *
00016  *  You should have received a copy of the GNU Lesser General Public
00017  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00018  ********************************************************************************
00019  */
00020
00021
00022
00023 /*
00024     d-bus interface for FunctionGenerator
00025     uses FunctionGeneratorImpl
00026 */
00027
00028
00029 #ifndef FUNCTION_GENERATOR_HPP_
00030 #define FUNCTION_GENERATOR_HPP_
00031
00032 #include "Owned.hpp"
00033 // @saftbus-export
00034 #include "Time.hpp"
00035
00036 #include <deque>
00037 #include "FunctionGeneratorImpl.hpp"
00038
00039 namespace saftlib {
00040
00041 // class TimingReceiver;
00042
00043
00074 class FunctionGenerator : public Owned
00075 {
00076
00077   public:
00078
00079     // FunctionGenerator(const ConstructorType& args);
00080     FunctionGenerator(saftbus::Container *container, const std::string &fg_name, const std::string
      &object_path, std::shared_ptr<FunctionGeneratorImpl> impl);
00081     ~FunctionGenerator();
00082
00083     // typedef FunctionGenerator_Service ServiceType;
00084     // struct ConstructorType {
00085     //     std::string objectPath;
00086     //     TimingReceiver *dev;
00087     //     std::shared_ptr<FunctionGeneratorImpl> functionGeneratorImpl;
00088     // };
00089
00090     // static std::shared_ptr<FunctionGenerator> create(saftbus::Container *container, const
      std::string &object_path, TimingReceiver *timing_receiver, std::shared_ptr<FunctionGeneratorImpl>
      impl);
00091
00092
00102     // @saftbus-export
00103     void Arm();
00104
00117     // @saftbus-export
00118     void Abort();
00119
00128     // @saftbus-export
00129     uint64_t ReadFillLevel();
00130
00154     // @saftbus-export
00155     bool AppendParameterSet(const std::vector< int16_t >& coeff_a, const std::vector< int16_t >&
      coeff_b, const std::vector< int32_t >& coeff_c, const std::vector< unsigned char >& step, const
      std::vector< unsigned char >& freq, const std::vector< unsigned char >& shift_a, const std::vector<
      unsigned char >& shift_b);
00156
00162     // @saftbus-export
00163     void Flush();
00164
00167     // @saftbus-export
00168     uint32_t getVersion() const;
```

```
00169
00172     // @saftbus-export
00173     unsigned char getSCUbusSlot() const;
00174
00177     // @saftbus-export
00178     unsigned char getDeviceNumber() const;
00179
00182     // @saftbus-export
00183     unsigned char getOutputWindowSize() const;
00184
00192     // @saftbus-export
00193     bool getEnabled() const;
00194
00205     // @saftbus-export
00206     bool getArmed() const;
00207
00217     // @saftbus-export
00218     bool getRunning() const;
00219
00220
00228     // @saftbus-export
00229     uint32_t getStartTag() const;
00230     // @saftbus-export
00231     void setStartTag(uint32_t val);
00232
00233
00241     // @saftbus-export
00242     uint32_t ReadExecutedParameterCount();
00243
00244
00245
00246     // Signals
00253     // @saftbus-export
00254     sigc::signal< void , bool > SigEnabled;
00255
00264     // @saftbus-export
00265     sigc::signal< void , bool > SigArmed;
00266
00275     // @saftbus-export
00276     sigc::signal< void , bool > SigRunning;
00277
00284     // @saftbus-export
00285     sigc::signal< void , saftlib::Time > SigStarted;
00286
00311     // @saftbus-export
00312     sigc::signal< void , saftlib::Time , bool , bool , bool > SigStopped;
00313
00323     // @saftbus-export
00324     sigc::signal< void > Refill;
00325
00326
00327     std::string getObjectPath();
00328
00329   protected:
00330     void Reset();
00331     void ownerQuit();
00332
00333     TimingReceiver *dev;
00334
00335     void on_fg_running(bool);
00336     void on_fg_armed(bool);
00337     void on_fg_enabled(bool);
00338     void on_fg_refill();
00339     void on_fg_started(uint64_t);
00340     void on_fg_stopped(uint64_t, bool, bool, bool);
00341
00342
00343     std::string name;
00344     std::string objectPath;
00345     std::shared_ptr<FunctionGeneratorImpl> fgImpl;
00346
00347 };
00348
00349 }
00350
00351 #endif
```

8.59 FunctionGenerator_Proxy.hpp

```
00001 #ifndef FunctionGenerator_PROXY_HPP_
00002 #define FunctionGenerator_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
```

```
00005
00006 #include <sigc++/sigc++.h>
00007
00008 #include "Time.hpp"
00009
00010 #include "Owned_Proxy.hpp"
00011
00012 namespace saftlib {
00013
00044     class FunctionGenerator_Proxy : public  Owned_Proxy   {
00045         static std::vector<std::string> gen_interface_names();
00046     public:
00047         FunctionGenerator_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00048         static std::shared_ptr<FunctionGenerator_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00049         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00059     // @saftbus-export
00060         void Arm();
00073     // @saftbus-export
00074         void Abort();
00083     // @saftbus-export
00084         uint64_t ReadFillLevel();
00108     // @saftbus-export
00109         bool AppendParameterSet(const std::vector< int16_t >& coeff_a, const std::vector< int16_t >&
      coeff_b, const std::vector< int32_t >& coeff_c, const std::vector< unsigned char >& step, const
      std::vector< unsigned char >& freq, const std::vector< unsigned char >& shift_a, const std::vector<
      unsigned char >& shift_b);
00115     // @saftbus-export
00116         void Flush();
00119     // @saftbus-export
00120         uint32_t getVersion();
00123     // @saftbus-export
00124         unsigned char getSCUbusSlot();
00127     // @saftbus-export
00128         unsigned char getDeviceNumber();
00131     // @saftbus-export
00132         unsigned char getOutputWindowSize();
00140     // @saftbus-export
00141         bool getEnabled();
00152     // @saftbus-export
00153         bool getArmed();
00163     // @saftbus-export
00164         bool getRunning();
00172     // @saftbus-export
00173         uint32_t getStartTag();
00174     // @saftbus-export
00175         void setStartTag(uint32_t val);
00183     // @saftbus-export
00184         uint32_t ReadExecutedParameterCount();
00185     // Signals
00192     // @saftbus-export
00193         sigc::signal<void, bool> SigEnabled;
00202     // @saftbus-export
00203         sigc::signal<void, bool> SigArmed;
00212     // @saftbus-export
00213         sigc::signal<void, bool> SigRunning;
00220     // @saftbus-export
00221         sigc::signal<void, saftlib::Time> SigStarted;
00246     // @saftbus-export
00247         sigc::signal<void, saftlib::Time, bool, bool, bool> SigStopped;
00257     // @saftbus-export
00258         sigc::signal<void> Refill;
00259     private:
00260         int interface_no;
00261
00262     };
00263
00264 }
00265
00266 #endif
```

8.60 FunctionGenerator_Service.hpp

```
00001 #ifndef FunctionGenerator_SERVICE_HPP_
00002 #define FunctionGenerator_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
```

```
00009 namespace saftlib {
00010
00011     class FunctionGenerator;
00012     class FunctionGenerator_Service : public saftbus::Service {
00013     public:
00014         FunctionGenerator* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef FunctionGenerator DriverType;
00017         FunctionGenerator_Service(FunctionGenerator* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         FunctionGenerator_Service();
00019         ~FunctionGenerator_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void SigEnabled_dispatch_function(bool arg_1);
00023         sigc::connection SigEnabled_connection;
00024         void SigArmed_dispatch_function(bool arg_1);
00025         sigc::connection SigArmed_connection;
00026         void SigRunning_dispatch_function(bool arg_1);
00027         sigc::connection SigRunning_connection;
00028         void SigStarted_dispatch_function(saftlib::Time arg_1);
00029         sigc::connection SigStarted_connection;
00030         void SigStopped_dispatch_function(saftlib::Time arg_1,bool arg_2,bool arg_3,bool arg_4);
00031         sigc::connection SigStopped_connection;
00032         void Refill_dispatch_function();
00033         sigc::connection Refill_connection;
00034         void Destroyed_dispatch_function();
00035         sigc::connection Destroyed_connection;
00036     };
00037
00038 }
00039
00040 #endif
```

8.61 FunctionGeneratorFirmware.hpp

```
00001 /*  Copyright (C) 2011-2016 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *
00005  ********************************************************************************
00006  *  This library is free software; you can redistribute it and/or
00007  *  modify it under the terms of the GNU Lesser General Public
00008  *  License as published by the Free Software Foundation; either
00009  *  version 3 of the License, or (at your option) any later version.
00010  *
00011  *  This library is distributed in the hope that it will be useful,
00012  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  *  Lesser General Public License for more details.
00015  *
00016  *  You should have received a copy of the GNU Lesser General Public
00017  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00018  ********************************************************************************
00019  */
00020
00021 #ifndef FUNCTION_GENERATOR_FIRMWARE_HPP_
00022 #define FUNCTION_GENERATOR_FIRMWARE_HPP_
00023
00024 #include "Owned.hpp"
00025 #include <TimingReceiver.hpp>
00026 #include "FunctionGenerator.hpp"
00027 #include "MasterFunctionGenerator.hpp"
00028
00029 namespace saftlib {
00030
00031 class TimingReceiver;
00032
00033
00038 class FunctionGeneratorFirmware : public Owned, public TimingReceiverAddon
00039 {
00040
00041   public:
00042
00043     std::map< std::string , std::map<std::string, std::string> > getObjects(); // TimingReceiverAddon
      pure-virtual override
00044     std::string getObjectPath(); // used in create_service.cpp
00045
00046
00047     FunctionGeneratorFirmware(saftbus::Container *container, SAFTd *saft_daemon, TimingReceiver
      *timing_receiver);
00048     ~FunctionGeneratorFirmware();
```

```
00049     // typedef FunctionGeneratorFirmware_Service ServiceType;
00050     // struct ConstructorType {
00051     //   std::string objectPath;
00052     //   TimingReceiver  *tr;
00053     //   Device &device;
00054     //   etherbone::sdb_msi_device  sdb_msi_base;
00055     //   sdb_device                 mailbox;
00056     //   std::map< std::string, std::shared_ptr<Owned> > &fgs_owned;
00057     //   std::map< std::string, std::shared_ptr<Owned> > &master_fgs_owned;
00058     // };
00059     // static std::shared_ptr<FunctionGeneratorFirmware> create(const ConstructorType& args);
00060
00061     uint32_t getVersion() const;
00062
00069     // @saftbus-export
00070     std::map<std::string, std::string> Scan();
00071     // @saftbus-export
00072     std::map<std::string, std::string> ScanMasterFg();
00073     // @saftbus-export
00074     std::map<std::string, std::string> ScanFgChannels();
00075
00076     void removeMasterFunctionGenerator();
00077
00078   protected:
00079
00081     void clear();
00082
00083     void firmware_rescan(int host_to_lm32_mailbox_slot_idx);
00084
00085     bool nothing_runs();
00086
00087     saftbus::Container         *container;
00088
00089     std::string                objectPath;
00090     SAFTd                      *saftd;
00091     TimingReceiver             *tr;
00092     Mailbox                    *mbox;
00093     etherbone::Device&         device;
00094     // etherbone::sdb_msi_device  sdb_msi_base;
00095     // sdb_device                 mailbox;
00096
00097     std::map< std::string, std::shared_ptr<FunctionGenerator> > fgs;
00098     std::shared_ptr<MasterFunctionGenerator> mfg;
00099
00100     bool have_fg_firmware;
00101     eb_data_t magic;
00102     eb_data_t version;
00103     eb_address_t fgb;
00104
00105     std::map<std::string, std::map<std::string, std::string> > addon_objects;
00106
00107
00108 };
00109
00110 }
00111
00112 #endif
```

8.62 FunctionGeneratorFirmware_Proxy.hpp

```
00001 #ifndef FunctionGeneratorFirmware_PROXY_HPP_
00002 #define FunctionGeneratorFirmware_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "Owned_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00016     class FunctionGeneratorFirmware_Proxy : public  Owned_Proxy   {
00017         static std::vector<std::string> gen_interface_names();
00018     public:
00019         FunctionGeneratorFirmware_Proxy(const std::string &object_path, saftbus::SignalGroup
      &signal_group = saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00020         static std::shared_ptr<FunctionGeneratorFirmware_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00021         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00028     // @saftbus-export
00029         std::map<std::string, std::string> Scan();
00030     // @saftbus-export
```

```
00031         std::map<std::string, std::string> ScanMasterFg();
00032     // @saftbus-export
00033         std::map<std::string, std::string> ScanFgChannels();
00034     private:
00035         int interface_no;
00036 
00037     };
00038 
00039 }
00040 
00041 #endif
```

8.63 FunctionGeneratorFirmware_Service.hpp

```
00001 #ifndef FunctionGeneratorFirmware_SERVICE_HPP_
00002 #define FunctionGeneratorFirmware_SERVICE_HPP_
00003 
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006 
00007 #include <functional>
00008 
00009 namespace saftlib {
00010 
00011     class FunctionGeneratorFirmware;
00012     class FunctionGeneratorFirmware_Service : public saftbus::Service {
00013     public:
00014         FunctionGeneratorFirmware* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef FunctionGeneratorFirmware DriverType;
00017         FunctionGeneratorFirmware_Service(FunctionGeneratorFirmware* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         FunctionGeneratorFirmware_Service();
00019         ~FunctionGeneratorFirmware_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021 
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025 
00026 }
00027 
00028 #endif
```

8.64 FunctionGeneratorImpl.hpp

```
00001 /*  Copyright (C) 2011-2016 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *
00005  ********************************************************************************
00006  *  This library is free software; you can redistribute it and/or
00007  *  modify it under the terms of the GNU Lesser General Public
00008  *  License as published by the Free Software Foundation; either
00009  *  version 3 of the License, or (at your option) any later version.
00010  *
00011  *  This library is distributed in the hope that it will be useful,
00012  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  *  Lesser General Public License for more details.
00015  *
00016  *  You should have received a copy of the GNU Lesser General Public
00017  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00018  ********************************************************************************
00019  */
00020 #ifndef FUNCTION_GENERATOR_IMPL_HPP_
00021 #define FUNCTION_GENERATOR_IMPL_HPP_
00022 
00023 #define ETHERBONE_THROWS 1
00024 #define __STDC_FORMAT_MACROS
00025 #define __STDC_CONSTANT_MACROS
00026 #include <etherbone.h>
00027 
00028 
00029 #include <deque>
00030 #include <memory>
00031 #include <boost/circular_buffer.hpp>
00032 #include <sigc++/sigc++.h>
```

```
00033 #include "Time.hpp"
00034 #include "Mailbox.hpp"
00035
00036 #include <saftbus/loop.hpp>
00037
00038
00039 namespace saftlib {
00040
00041 class SAFTd;
00042 class IRQ;
00043 class TimingReceiver;
00044 // class Mailbox;
00045
00046 // class FunctionGeneratorChannelAllocation //: public Glib::Object
00047 // {
00048 //   public:
00049 //     std::vector<int> indexes;
00050 // };
00051
00052   struct ParameterTuple {
00053       int16_t coeff_a;
00054       int16_t coeff_b;
00055       int32_t coeff_c;
00056       uint8_t step;
00057       uint8_t freq;
00058       uint8_t shift_a;
00059       uint8_t shift_b;
00060
00061       uint64_t duration() const;
00062     };
00063
00064 class FunctionGeneratorImpl //: public Glib::Object
00065 {
00066     friend class MasterFunctionGenerator;
00067
00068   public:
00069 //    typedef FunctionGenerator_Service ServiceType;
00070     // struct ConstructorType {
00071     //   std::string objectPath;
00072     //   TimingReceiver *tr;
00073     //   std::shared_ptr<std::vector<int> > allocation;
00074     //   eb_address_t fgb;
00075     //   eb_address_t swi;
00076     //   etherbone::sdb_msi_device base;
00077     //   sdb_device mbx;
00078     //   unsigned num_channels;
00079     //   unsigned buffer_size;
00080     //   unsigned int index;
00081     //   uint32_t macro;
00082     // };
00083
00084     FunctionGeneratorImpl(SAFTd *saft_daemon
00085                         , TimingReceiver *timing_receiver
00086                         , std::shared_ptr<std::vector<int> > channel_allocation
00087                         , eb_address_t fgb
00088                         , int mbox_slot
00089                         , unsigned n_channels
00090                         , unsigned buf_size
00091                         , unsigned idx
00092                         , uint32_t macro);
00093     // FunctionGeneratorImpl(const ConstructorType& args);
00094     ~FunctionGeneratorImpl();
00095
00096     //static std::shared_ptr<FunctionGenerator> create(const ConstructorType& args);
00097
00098     template<typename Iter> bool appendParameterTuples(Iter it, Iter end)
00099     {
00100       for (; it != end; ++it)
00101       {
00102         fifo_push_back(*it);
00103         fillLevel += (*it).duration();
00104       }
00105
00106       if (channel != -1) refill(false);
00107       return lowFill();
00108     }
00109
00110     bool appendParameterTuples(std::vector<ParameterTuple> parameters);
00111
00112
00113     void Arm();
00114     void Abort();
00115     uint64_t ReadFillLevel();
00116     bool appendParameterSet(const std::vector< int16_t >& coeff_a, const std::vector< int16_t >&
      coeff_b, const std::vector< int32_t >& coeff_c, const std::vector< unsigned char >& step, const
      std::vector< unsigned char >& freq, const std::vector< unsigned char >& shift_a, const std::vector<
      unsigned char >& shift_b);
```

```
00117     void Flush();
00118     uint32_t getVersion() const;
00119     unsigned char getSCUbusSlot() const;
00120     unsigned char getDeviceNumber() const;
00121     unsigned char getOutputWindowSize() const;
00122     bool getEnabled() const;
00123     bool getArmed() const;
00124     bool getRunning() const;
00125     uint32_t getStartTag() const;
00126     uint32_t ReadExecutedParameterCount();
00127     void setStartTag(uint32_t val);
00128 
00129     std::string GetName();
00130 
00131     sigc::signal<void, bool> signal_enabled;
00132     sigc::signal<void, bool> signal_running;
00133     sigc::signal<void, bool> signal_armed;
00134     sigc::signal<void> signal_refill;
00135     sigc::signal<void, uint64_t> signal_started;
00136     sigc::signal<void, uint64_t, bool, bool, bool> signal_stopped;
00137 
00138     void flush();
00139     void arm();
00140     void Reset();
00141 
00142 
00143 
00144   protected:
00145     bool lowFill() const;
00146     void irq_handler(eb_data_t msi);
00147     void refill(bool);
00148     void releaseChannel();
00149     void acquireChannel();
00150 
00151     bool ResetFailed();
00152     void ownerQuit();
00153 
00154     void fifo_push_back(const ParameterTuple& tuple);
00155 
00156 
00157 
00158     SAFTd          *saftd;
00159     TimingReceiver *tr;
00160     Mailbox        *mbx;
00161     etherbone::Device &device;
00162     std::shared_ptr<std::vector<int> > allocation;
00163     eb_address_t shm;
00164     // eb_address_t swi;
00165     // etherbone::sdb_msi_device base;
00166     // sdb_device mbx;
00167     int mb_slot; // host->lm32 mailbox slot (owned by lm32, therefore we have only the index)
00168     unsigned num_channels;
00169     unsigned buffer_size;
00170     unsigned int index;
00171     unsigned char scubusSlot;
00172     unsigned char deviceNumber;
00173     unsigned char version;
00174     unsigned char outputWindowSize;
00175     std::unique_ptr<saftlib::IRQ> irq;
00176     std::unique_ptr<Mailbox::Slot> host_slot; // lm32->host (owned by host, therefore we have a
      Mailbox::Slot object)
00177 
00178     int channel; // -1 if no channel assigned
00179     bool enabled;
00180     bool armed;
00181     bool running;
00182     bool abort;
00183 
00184     //sigc::connection resetTimeout;
00185     saftbus::SourceHandle resetTimeout;
00186     uint32_t startTag;
00187     unsigned executedParameterCount;
00188 
00189     unsigned mbx_slot;
00190     eb_address_t mailbox_slot_address;
00191 
00192     // These 3 variables must be kept in sync:
00193     uint64_t fillLevel;
00194     unsigned filled; // # of fifo entries currently on LM32
00195     boost::circular_buffer<ParameterTuple> fifo;
00196 
00197     unsigned fg_fifo_max_size;
00198 };
00199 
00200 }
00201 
00202 #endif
```

8.65 Input.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef saftlib_INPUT_HPP_
00023 #define saftlib_INPUT_HPP_
00024
00025 #include "Io.hpp"
00026 #include "ECA.hpp"
00027 #include "ECA_TLU.hpp"
00028 #include "EventSource.hpp"
00029
00030 namespace saftlib {
00031
00033 class Input : public EventSource
00034 {
00035     public:
00036
00037         Input(ECA_TLU &eca_tlu
00038             , const std::string &input_object_path
00039             , const std::string &output_partner_path
00040             , unsigned eca_tlu_chan
00041             , Io *io
00042             , saftbus::Container *container = nullptr);
00043
00044         // Methods
00052         // @saftbus-export
00053         bool ReadInput();
00054         // Property getters
00055
00064         // @saftbus-export
00065         uint32_t getStableTime() const;
00068         // @saftbus-export
00069         void setStableTime(uint32_t val);
00070
00073         // @saftbus-export
00074         uint32_t getIndexIn() const;
00075
00078         // @saftbus-export
00079         bool getSpecialPurposeIn() const;
00081         // @saftbus-export
00082         void setSpecialPurposeIn(bool val);
00085         // @saftbus-export
00086         bool getSpecialPurposeInAvailable() const;
00087
00090         // @saftbus-export
00091         bool getGateIn() const;
00094         // @saftbus-export
00095         void setGateIn(bool val);
00096
00097
00105         // @saftbus-export
00106         bool getInputTermination() const;
00109         // @saftbus-export
00110         void setInputTermination(bool val);
00113         // @saftbus-export
00114         bool getInputTerminationAvailable() const;
00115
00116
00119         // @saftbus-export
00120         std::string getLogicLevelIn() const;
00121
00124         // @saftbus-export
00125         std::string getTypeIn() const;
00126
00129         // @saftbus-export
00130         std::string getOutput() const;
00131         // Property setters
```

```
00132
00133         // From iEventSource
00134         uint64_t getResolution() const;
00135         uint32_t getEventBits() const;
00136         bool getEventEnable() const;
00137         uint64_t getEventPrefix() const;
00138
00139         void setEventEnable(bool val);
00140         void setEventPrefix(uint64_t val);
00141         //  sigc::signal< void, bool > EventEnable;
00142         //  sigc::signal< void, uint64_t > EventPrefix;
00143
00144     private:
00145         ECA_TLU &eca_tlu;
00146         Io *io;
00147         std::string partnerPath;
00148         eb_address_t tlu;
00149         unsigned eca_tlu_channel;
00150         bool enable;
00151         uint64_t event;
00152         uint32_t stable;
00153
00154         // void configInput();
00155 };
00156
00157 }
00158
00159 #endif /* INPUT_H */
```

8.66 Input_Proxy.hpp

```
00001 #ifndef Input_PROXY_HPP_
00002 #define Input_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "EventSource_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00013     class Input_Proxy : public  EventSource_Proxy   {
00014         static std::vector<std::string> gen_interface_names();
00015     public:
00016         Input_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00017         static std::shared_ptr<Input_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00018         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00019         // Methods
00027         // @saftbus-export
00028         bool ReadInput();
00037         // @saftbus-export
00038         uint32_t getStableTime();
00041         // @saftbus-export
00042         void setStableTime(uint32_t val);
00045         // @saftbus-export
00046         uint32_t getIndexIn();
00049         // @saftbus-export
00050         bool getSpecialPurposeIn();
00052         // @saftbus-export
00053         void setSpecialPurposeIn(bool val);
00056         // @saftbus-export
00057         bool getSpecialPurposeInAvailable();
00060         // @saftbus-export
00061         bool getGateIn();
00064         // @saftbus-export
00065         void setGateIn(bool val);
00073         // @saftbus-export
00074         bool getInputTermination();
00077         // @saftbus-export
00078         void setInputTermination(bool val);
00081         // @saftbus-export
00082         bool getInputTerminationAvailable();
00085         // @saftbus-export
00086         std::string getLogicLevelIn();
00089         // @saftbus-export
00090         std::string getTypeIn();
00093         // @saftbus-export
00094         std::string getOutput();
00095     private:
```

```
00096         int interface_no;
00097
00098     };
00099
00100 }
00101
00102 #endif
```

8.67 Input_Service.hpp

```
00001 #ifndef Input_SERVICE_HPP_
00002 #define Input_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class Input;
00012     class Input_Service : public saftbus::Service {
00013     public:
00014         Input* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef Input DriverType;
00017         Input_Service(Input* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         Input_Service();
00019         ~Input_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.68 CommonFunctions.h

```
00001 #ifndef COMMON_FUNCTIONS_COMPATIBILITY_H_
00002 #define COMMON_FUNCTIONS_COMPATIBILITY_H_
00003
00004 #include <CommonFunctions.hpp>
00005
00006 #endif
```

8.69 CommonFunctions.h

```
00001 #ifndef COMMON_FUNCTIONS_COMPATIBILITY_H_
00002 #define COMMON_FUNCTIONS_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <CommonFunctions.hpp>
00006
00007 #endif
```

8.70 EmbeddedCPUActionSink.h

```
00001 #ifndef EMBEDDEDCPUACTIONSINK_COMPATIBILITY_H_
00002 #define EMBEDDEDCPUACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <EmbeddedCPUActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.71 EmbeddedCPUCondition.h

```
00001 #ifndef EMBEDDEDCPUCONDITION_COMPATIBILITY_H_
00002 #define EMBEDDEDCPUCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <EmbeddedCPUCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.72 iActionSink.h

```
00001 #ifndef IACTIONSINK_COMPATIBILITY_H_
00002 #define IACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <ActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.73 iCondition.h

```
00001 #ifndef ICONDITION_COMPATIBILITY_H_
00002 #define ICONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <Condition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.74 iDevice.h

```
00001 #ifndef IDEVICE_COMPATIBILITY_H_
00002 #define IDEVICE_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <OpenDevice_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.75 iEmbeddedCPUActionSink.h

```
00001 #ifndef IEMBEDDEDCPUACTIONSINK_COMPATIBILITY_H_
00002 #define IEMBEDDEDCPUACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <EmbeddedCPUActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.76 iEmbeddedCPUCondition.h

```
00001 #ifndef IEMBEDDEDCPUCONDITION_COMPATIBILITY_H_
00002 #define IEMBEDDEDCPUCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <EmbeddedCPUCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.77 iEventSource.h

```
00001 #ifndef IEVENTSOURCE_COMPATIBILITY_H_
00002 #define IEVENTSOURCE_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <EventSource_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.78 iInputEventSource.h

```
00001 #ifndef IINPUTEVENTSOURCE_COMPATIBILITY_H_
00002 #define IINPUTEVENTSOURCE_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <InputEventSource_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.79 iMILbusActionSink.h

```
00001 #ifndef IMILBUSACTIONSINK_COMPATIBILITY_H_
00002 #define IMILBUSACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <MILbusActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.80 iMILbusCondition.h

```
00001 #ifndef IMILBUSCONDITION_COMPATIBILITY_H_
00002 #define IMILBUSCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <MILbusCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.81 Input.h

```
00001 #ifndef INPUT_COMPATIBILITY_H_
00002 #define INPUT_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <Input_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.82 iOutputActionSink.h

```
00001 #ifndef IOUTPUTACTIONSINK_COMPATIBILITY_H_
00002 #define IOUTPUTACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <OutputActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.83 iOutputCondition.h

```
00001 #ifndef IOUTPUTCONDITION_COMPATIBILITY_H_
00002 #define IOUTPUTCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <OutputCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.84 iOwned.h

```
00001 #ifndef IOWNED_COMPATIBILITY_H_
00002 #define IOWNED_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <Owned_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.85 iSAFTd.h

```
00001 #ifndef ISAFTD_COMPATIBILITY_H_
00002 #define ISAFTD_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SAFTd_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.86 iSCUbusActionSink.h

```
00001 #ifndef ISCUBUSACTIONSINK_COMPATIBILITY_H_
00002 #define ISCUBUSACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SCUbusActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.87 iSCUbusCondition.h

```
00001 #ifndef ISCUBUSCONDITION_COMPATIBILITY_H_
00002 #define ISCUBUSCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SCUbusCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.88 iSoftwareActionSink.h

```
00001 #ifndef ISOFTWAREACTIONSINK_COMPATIBILITY_H_
00002 #define ISOFTWAREACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SoftwareActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.89 iSoftwareCondition.h

```
00001 #ifndef ISOFTWARECONDITION_COMPATIBILITY_H_
00002 #define ISOFTWARECONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SoftwareCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.90 iTimingReceiver.h

```
00001 #ifndef ITIMINGRECEIVER_COMPATIBILITY_H_
00002 #define ITIMINGRECEIVER_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <TimingReceiver_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.91 iWbmActionSink.h

```
00001 #ifndef IWBMACTIONSINK_COMPATIBILITY_H_
00002 #define IWBMACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <WbmActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.92 iWbmCondition.h

```
00001 #ifndef IWBMCONDITION_COMPATIBILITY_H_
00002 #define IWBMCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <WbmCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.93 iWrMilGateway.h

```
00001 #ifndef IWRMILGATEWAY_COMPATIBILITY_H_
00002 #define IWRMILGATEWAY_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <WrMilGateway_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.94 MILbusActionSink.h

```
00001 #ifndef MILBUSACTIONSINK_COMPATIBILITY_H_
00002 #define MILBUSACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <MILbusActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.95 MILbusCondition.h

```
00001 #ifndef MILBUSCONDITION_COMPATIBILITY_H_
00002 #define MILBUSCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <MILbusCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.96 Output.h

```
00001 #ifndef OUTPUT_COMPATIBILITY_H_
00002 #define OUTPUT_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <Output_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.97 OutputCondition.h

```
00001 #ifndef OUTPUTCONDITION_COMPATIBILITY_H_
00002 #define OUTPUTCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <OutputCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.98 saftbus.h

```
00001 #ifndef SAFTBUS_COMPATIBILITY_H_
00002 #define SAFTBUS_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <saftbus/saftbus.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.99 SAFTd.h

```
00001 #ifndef SAFTD_COMPATIBILITY_H_
00002 #define SAFTD_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SAFTd_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.100 saftlib.h

```
00001 #ifndef SAFTLIB_COMPATIBILITY_H_
00002 #define SAFTLIB_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <saftbus/saftbus.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.101 SCUbusActionSink.h

```
00001 #ifndef SCUBUSACTIONSINK_COMPATIBILITY_H_
00002 #define SCUBUSACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SCUbusActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.102 SCUbusCondition.h

```
00001 #ifndef SCUBUSCONDITION_COMPATIBILITY_H_
00002 #define SCUBUSCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SCUbusCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.103 SoftwareActionSink.h

```
00001 #ifndef SOFTWAREACTIONSINK_COMPATIBILITY_H_
00002 #define SOFTWAREACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SoftwareActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.104 SoftwareCondition.h

```
00001 #ifndef SOFTWARECONDITION_COMPATIBILITY_H_
00002 #define SOFTWARECONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <SoftwareCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.105 TimingReceiver.h

```
00001 #ifndef TIMINGRECEIVER_COMPATIBILITY_H_
00002 #define TIMINGRECEIVER_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <TimingReceiver_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.106 WbmActionSink.h

```
00001 #ifndef WBMACTIONSINK_COMPATIBILITY_H_
00002 #define WBMACTIONSINK_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <WbmActionSink_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.107 WbmCondition.h

```
00001 #ifndef WBMCONDITION_COMPATIBILITY_H_
00002 #define WBMCONDITION_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <WbmCondition_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.108 WrMilGateway.h

```
00001 #ifndef WRMILGATEWAY_COMPATIBILITY_H_
00002 #define WRMILGATEWAY_COMPATIBILITY_H_
00003
00004 #include <saftbus/error.hpp>
00005 #include <WrMilGateway_Proxy.hpp>
00006 #include <CommonFunctions.hpp>
00007
00008 #endif
```

8.109 Io.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_IO_HPP_
00023 #define saftlib_IO_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include <functional>
00033 #include <string>
00034
00035 namespace saftlib {
00036
00037 class SerdesClockGen;
00038
00046 class Io {
00047     etherbone::Device &device;
00048     std::string io_name;
00049     unsigned io_direction;
00050     unsigned io_channel;
00051     unsigned io_eca_in;
00052     unsigned io_eca_out;
00053     unsigned io_index;
00054     unsigned io_special_purpose;
00055     unsigned io_logic_level;
00056     bool io_oe_available;
00057     bool io_term_available;
00058     bool io_spec_out_available;
00059     bool io_spec_in_available;
00060     eb_address_t io_control_addr;
00061     SerdesClockGen &io_clkgen;
00062 public:
```

```
00063     Io(etherbone::Device &device
00064        , const std::string &io_name
00065        , unsigned io_direction
00066        , unsigned io_channel
00067        , unsigned eca_in
00068        , unsigned eca_out
00069        , unsigned io_index
00070        , unsigned io_special_purpose
00071        , unsigned io_logic_level
00072        , bool io_oe_available
00073        , bool io_term_available
00074        , bool io_spec_out_available
00075        , bool io_spec_in_available
00076        , eb_address_t io_control_addr
00077        , SerdesClockGen &clkgen
00078     );
00079
00080     const std::string &getName() const;
00081     unsigned getDirection() const;
00082     // iOutputActionSink
00083     uint32_t getIndexOut() const;
00084     uint32_t getIndexIn() const;
00085     uint32_t getEcaIn() const;
00086     uint32_t getEcaOut() const;
00087     void WriteOutput(bool value);
00088     bool ReadOutput();
00089     bool ReadCombinedOutput();
00090     bool getOutputEnable() const;
00091     void setOutputEnable(bool val);
00092     bool getOutputEnableAvailable() const;
00093     bool getSpecialPurposeOut() const;
00094     void setSpecialPurposeOut(bool val);
00095     bool getSpecialPurposeOutAvailable() const;
00096     bool getGateOut() const;
00097     void setGateOut(bool val);
00098     bool getBuTiSMultiplexer() const;
00099     void setBuTiSMultiplexer(bool val);
00100     bool getPPSMultiplexer() const;
00101     void setPPSMultiplexer(bool val);
00102     bool StartClock(double high_phase, double low_phase, uint64_t phase_offset);
00103     bool StopClock();
00104     std::string getLogicLevelOut() const;
00105     std::string getTypeOut() const;
00106
00107     // iInputEventSource
00108     bool ReadInput(); // done
00109     bool getInputTermination() const;
00110     void setInputTermination(bool val);
00111     bool getInputTerminationAvailable() const;
00112     bool getSpecialPurposeIn() const;
00113     void setSpecialPurposeIn(bool val);
00114     bool getSpecialPurposeInAvailable() const;
00115     bool getGateIn() const;
00116     void setGateIn(bool val);
00117     std::string getLogicLevelIn() const;
00118     std::string getTypeIn() const;
00119
00120     // iInputEventSource
00121     uint64_t getResolution() const;
00122
00123     bool ConfigureClock(double high_phase, double low_phase, uint64_t phase_offset);
00124     std::string getLogicLevel() const;
00125     std::string getType() const;
00126
00127
00128     std::function< void(bool) > OutputEnable;
00129     std::function< void(bool) > SpecialPurposeOut;
00130     std::function< void(bool) > GateOut;
00131     std::function< void(bool) > BuTiSMultiplexer;
00132     std::function< void(bool) > PPSMultiplexer;
00133     std::function< void(bool) > InputTermination;
00134     std::function< void(bool) > SpecialPurposeIn;
00135     std::function< void(bool) > GateIn;
00136
00137 };
00138
00139 }
00140
00141 #endif
```

8.110 io_control_regs.h

```
00001
```

```
00020 #ifndef IOCONTROL_REGS_H
00021 #define IOCONTROL_REGS_H
00022
00023 /* Defines */
00024 /*
      ============================================================================================================
      */
00025 #define IO_CONTROL_PRODUCT_ID        0x10c05791
00026 #define IO_CONTROL_VENDOR_ID         0x00000651
00027
00028 #define IO_MAX_IOS_PER_CHANNEL       64 /* 64 IOs per channel */
00029 #define IO_MAX_VALID_CHANNELS        2  /* GPIO and LVDS */
00030
00031 #define IO_LVDS_MAX                  64
00032 #define IO_GPIO_MAX                  64
00033 #define IO_FIXED_MAX                 64
00034
00035 #define IO_LEGACY_MODE_ENABLE        0x1
00036 #define IO_LEGACY_MODE_DISABLE       0x0
00037
00038 #define IO_INFO_INOUT_COUNT_MASK     0x000000ff
00039 #define IO_INFO_OUT_COUNT_MASK       0x0000ff00
00040 #define IO_INFO_IN_COUNT_MASK        0x00ff0000
00041 #define IO_INFO_TOTAL_COUNT_MASK     0xff000000
00042 #define IO_INFO_INOUT_SHIFT          0
00043 #define IO_INFO_OUT_SHIFT            8
00044 #define IO_INFO_IN_SHIFT             16
00045 #define IO_INFO_TOTAL_SHIFT          24
00046
00047 #define IO_FIELD_PARAM_INVALID       0xffffffff
00048 #define IO_FIELD_NUNBER_MASK         0xff000000
00049 #define IO_FIELD_INTERNAL_ID_MASK    0x00ff0000
00050 #define IO_FIELD_CFG_MASK            0x0000ff00
00051 #define IO_FIELD_LOGIC_RES_MASK      0x000000ff
00052 #define IO_FIELD_NUNBER_SHIFT        24
00053 #define IO_FIELD_INTERNAL_ID_SHIFT   16
00054 #define IO_FIELD_CFG_SHIFT           8
00055 #define IO_FIELD_LOGIC_RES_SHIFT     0
00056
00057 #define IO_SPECIAL_PURPOSE_MASK      0xfc
00058 #define IO_SPECIAL_OUT_MASK          0x02
00059 #define IO_SPECIAL_IN_MASK           0x01
00060
00061 #define IO_SPECIAL_PURPOSE_SHIFT     2
00062 #define IO_SPECIAL_OUT_SHIFT         1
00063 #define IO_SPECIAL_IN_SHIFT          0
00064
00065 #define IO_SPECIAL_MTCA4_BPL_BUF_OE    6
00066 #define IO_SPECIAL_LIBERA_TRIG_OE      5
00067 #define IO_SPECIAL_MTCA4_FAILSAFE_EN   4
00068 #define IO_SPECIAL_MTCA4_TRIG_BPL_PDN  3
00069 #define IO_SPECIAL_CLK_IN_EN           2
00070 #define IO_SPECIAL_TTL_TO_NIM          1
00071 #define IO_SPECIAL_NONE                0
00072
00073 #define IO_CFG_FIELD_DIR_MASK        0xc0
00074 #define IO_CFG_FIELD_INFO_CHAN_MASK  0x38
00075 #define IO_CFG_FIELD_OE_MASK         0x04
00076 #define IO_CFG_FIELD_TERM_MASK       0x02
00077 #define IO_CFG_FIELD_RES_BIT_MASK    0x01
00078
00079 #define IO_CFG_FIELD_DIR_SHIFT       6
00080 #define IO_CFG_FIELD_INFO_CHAN_SHIFT 3
00081 #define IO_CFG_FIELD_OE_SHIFT        2
00082 #define IO_CFG_FIELD_TERM_SHIFT      1
00083 #define IO_CFG_FIELD_RES_BIT_SHIFT   0
00084
00085 #define IO_LOGIC_RES_FIELD_LL_MASK   0xf0
00086 #define IO_LOGIC_RES_FIELD_RES_MASK  0x0f
00087
00088 #define IO_LOGIC_RES_FIELD_LL_SHIFT  4
00089 #define IO_LOGIC_RES_FIELD_RES_SHIFT 0
00090
00091 #define IO_CFG_FIELD_DIR_OUTPUT      0
00092 #define IO_CFG_FIELD_DIR_INPUT       1
00093 #define IO_CFG_FIELD_DIR_INOUT       2
00094
00095 #define IO_CFG_OE_UNAVAILABLE        0
00096 #define IO_CFG_OE_AVAILABLE          1
00097
00098 #define IO_CFG_TERM_UNAVAILABLE      0
00099 #define IO_CFG_TERM_AVAILABLE        1
00100
00101 #define IO_CFG_SPEC_UNAVAILABLE      0
00102 #define IO_CFG_SPEC_AVAILABLE        1
00103
00104 #define IO_CFG_CHANNEL_GPIO          0
```

```
00105 #define IO_CFG_CHANNEL_LVDS          1
00106 #define IO_CFG_CHANNEL_FIXED         2
00107
00108 #define IO_LOGIC_LEVEL_TTL           0
00109 #define IO_LOGIC_LEVEL_LVTTL         1
00110 #define IO_LOGIC_LEVEL_LVDS          2
00111 #define IO_LOGIC_LEVEL_NIM           3
00112 #define IO_LOGIC_LEVEL_CMOS          4
00113
00114 #define IO_OPERATION_SET             0
00115 #define IO_OPERATION_RESET           1
00116 #define IO_OPERATION_GET             2
00117
00118 #define IO_TYPE_OE                   0
00119 #define IO_TYPE_TERM                 1
00120 #define IO_TYPE_SPEC_IN              2
00121 #define IO_TYPE_SPEC_OUT             3
00122 #define IO_TYPE_MUX                  4
00123 #define IO_TYPE_SEL                  5
00124
00125 #define __IO_RETURN_FAILURE          1
00126 #define __IO_RETURN_SUCCESS          0
00127 #define __IO_RETURN_IO_NAME_UNKNOWN  0xffffffff
00128 #define __IO_RETURN_IO_OPTION_INVAL  0xfffffffe
00129
00130 /* Enumerations */
00131 /*
      ================================================================================================
      */
00132 typedef enum
00133 {
00134   eGPIO_Oe_Legacy_low        = 0x0000,
00135   eLVDS_Oe_Legacy_low        = 0x0004,
00136   eGPIO_Oe_Legacy_high       = 0x0008,
00137   eLVDS_Oe_Legacy_high       = 0x000c,
00138   eIO_Config                 = 0x0010,
00139   eIO_Version                = 0x0100,
00140   eGPIO_Info                 = 0x0104,
00141   eLVDS_Info                 = 0x0108,
00142   eFIXED_Info                = 0x010c,
00143   eGPIO_Oe_Set_low           = 0x0200,
00144   eGPIO_Oe_Set_high          = 0x0204,
00145   eGPIO_Oe_Reset_low         = 0x0208,
00146   eGPIO_Oe_Reset_high        = 0x020c,
00147   eLVDS_Oe_Set_low           = 0x0300,
00148   eLVDS_Oe_Set_high          = 0x0304,
00149   eLVDS_Oe_Reset_low         = 0x0308,
00150   eLVDS_Oe_Reset_high        = 0x030c,
00151   eGPIO_Term_Set_low         = 0x0400,
00152   eGPIO_Term_Set_high        = 0x0404,
00153   eGPIO_Term_Reset_low       = 0x0408,
00154   eGPIO_Term_Reset_high      = 0x040c,
00155   eLVDS_Term_Set_low         = 0x0500,
00156   eLVDS_Term_Set_high        = 0x0504,
00157   eLVDS_Term_Reset_low       = 0x0508,
00158   eLVDS_Term_Reset_high      = 0x050c,
00159   eGPIO_Spec_In_Set_low      = 0x0600,
00160   eGPIO_Spec_In_Set_high     = 0x0604,
00161   eGPIO_Spec_In_Reset_low    = 0x0608,
00162   eGPIO_Spec_In_Reset_high   = 0x060c,
00163   eGPIO_Spec_Out_Set_low     = 0x0700,
00164   eGPIO_Spec_Out_Set_high    = 0x0704,
00165   eGPIO_Spec_Out_Reset_low   = 0x0708,
00166   eGPIO_Spec_Out_Reset_high  = 0x070c,
00167   eLVDS_Spec_In_Set_low      = 0x0800,
00168   eLVDS_Spec_In_Set_high     = 0x0804,
00169   eLVDS_Spec_In_Reset_low    = 0x0808,
00170   eLVDS_Spec_In_Reset_high   = 0x080c,
00171   eLVDS_Spec_Out_Set_low     = 0x0900,
00172   eLVDS_Spec_Out_Set_high    = 0x0904,
00173   eLVDS_Spec_Out_Reset_low   = 0x0908,
00174   eLVDS_Spec_Out_Reset_high  = 0x090c,
00175   eGPIO_Mux_Set_low          = 0x0a00,
00176   eGPIO_Mux_Set_high         = 0x0a04,
00177   eGPIO_Mux_Reset_low        = 0x0a08,
00178   eGPIO_Mux_Reset_high       = 0x0a0c,
00179   eLVDS_Mux_Set_low          = 0x0b00,
00180   eLVDS_Mux_Set_high         = 0x0b04,
00181   eLVDS_Mux_Reset_low        = 0x0b08,
00182   eLVDS_Mux_Reset_high       = 0x0b0c,
00183   eGPIO_Sel_Set_low          = 0x0c00,
00184   eGPIO_Sel_Set_high         = 0x0c04,
00185   eGPIO_Sel_Reset_low        = 0x0c08,
00186   eGPIO_Sel_Reset_high       = 0x0c0c,
00187   eLVDS_Sel_Set_low          = 0x0d00,
00188   eLVDS_Sel_Set_high         = 0x0d04,
00189   eLVDS_Sel_Reset_low        = 0x0d08,
```

```
00190   eLVDS_Sel_Reset_high       = 0x0d0c,
00191   eGPIO_Gate_In_Set_low      = 0x1000,
00192   eGPIO_Gate_In_Set_high     = 0x1004,
00193   eGPIO_Gate_In_Reset_low    = 0x1008,
00194   eGPIO_Gate_In_Reset_high   = 0x100c,
00195   eLVDS_Gate_In_Set_low      = 0x2000,
00196   eLVDS_Gate_In_Set_high     = 0x2004,
00197   eLVDS_Gate_In_Reset_low    = 0x2008,
00198   eLVDS_Gate_In_Reset_high   = 0x200c,
00199   eGPIO_Gate_Out_Set_low     = 0x3000,
00200   eGPIO_Gate_Out_Set_high    = 0x3004,
00201   eGPIO_Gate_Out_Reset_low   = 0x3008,
00202   eGPIO_Gate_Out_Reset_high  = 0x300c,
00203   eLVDS_Gate_Out_Set_low     = 0x4000,
00204   eLVDS_Gate_Out_Set_high    = 0x4004,
00205   eLVDS_Gate_Out_Reset_low   = 0x4008,
00206   eLVDS_Gate_Out_Reset_high  = 0x400c,
00207   eSet_GPIO_Out_Begin        = 0xa000,
00208   eSet_LVDS_Out_Begin        = 0xb000,
00209   eGet_GPIO_In_Begin         = 0xc000,
00210   eGet_LVDS_In_Begin         = 0xd000,
00211   eGPIO_PPS_Mux_Set_low      = 0x0e00,
00212   eGPIO_PPS_Mux_Set_high     = 0x0e04,
00213   eGPIO_PPS_Mux_Reset_low    = 0x0e08,
00214   eGPIO_PPS_Mux_Reset_high   = 0x0e0c,
00215   eLVDS_PPS_Mux_Set_low      = 0x0f00,
00216   eLVDS_PPS_Mux_Set_high     = 0x0f04,
00217   eLVDS_PPS_Mux_Reset_low    = 0x0f08,
00218   eLVDS_PPS_Mux_Reset_high   = 0x0f0c,
00219   eIO_Map_Table_Begin        = 0xe000
00220 } e_IOCONTROL_RegisterArea;
00221
00222 /* Structures */
00223 /*
      ============================================================================================================
      */
00224 typedef struct
00225 {
00226   char          uName[12];
00227   unsigned char uSpecial;
00228   unsigned char uIndex;
00229   unsigned char uIOCfgSpace;
00230   unsigned char uLogicLevelRes;
00231 } s_IOCONTROL_SetupField;
00232
00233 #endif
```

8.111 loControl.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_IO_CONTROL_HPP_
00023 #define saftlib_IO_CONTROL_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "Io.hpp"
00033 #include "SerdesClockGen.hpp"
00034 #include "SdbDevice.hpp"
```

```
00035
00036 namespace saftlib {
00037
00039 class IoControl : public SdbDevice {
00040     SerdesClockGen clkgen;
00041
00042     std::vector<Io> ios;
00043 public:
00044     IoControl(etherbone::Device &device);
00045
00046     std::vector<Io> & get_ios();
00047 };
00048
00049 }
00050
00051 #endif
```

8.112 LM32Cluster.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef SAFTLIB_LM32CLUSTER_HPP_
00023 #define SAFTLIB_LM32CLUSTER_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include <vector>
00033 #include <string>
00034 #include <map>
00035 #include <memory>
00036 // #include <ltdl.h>
00037 #include <dlfcn.h>
00038
00039 // @saftbus-export
00040 #include <saftbus/service.hpp>
00041
00042 #include <SdbDevice.hpp>
00043
00044 namespace saftlib {
00045
00046 class TimingReceiver;
00047
00048 class LM32Cluster : public SdbDevice {
00049
00050     unsigned num_cores;
00051     unsigned ram_per_core;
00052     TimingReceiver *tr;
00053 public:
00054     LM32Cluster(etherbone::Device &device, TimingReceiver *tr);
00055     ~LM32Cluster();
00056
00057     std::vector<eb_address_t> dpram_lm32_adr_first;
00058     std::vector<eb_address_t> dpram_lm32_adr_last;
00059
00063     // @saftbus-export
00064     unsigned getCpuCount();
00065
00070     // @saftbus-export
00071     void SafeHaltCpu(unsigned cpu_idx);
```

```
00072
00073     void WriteFirmware(unsigned cpu_idx, const std::string &filename);
00074 };
00075
00076 }
00077
00078 #endif
```

8.113 LM32Cluster_Proxy.hpp

```
00001 #ifndef LM32Cluster_PROXY_HPP_
00002 #define LM32Cluster_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007 #include <saftbus/service.hpp>
00008
00009
00010 namespace saftlib {
00011
00012 // @saftbus-export
00013     class LM32Cluster_Proxy : public virtual saftbus::Proxy
00014     {
00015         static std::vector<std::string> gen_interface_names();
00016     public:
00017         LM32Cluster_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00018         static std::shared_ptr<LM32Cluster_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00019         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00023     // @saftbus-export
00024         unsigned getCpuCount();
00029     // @saftbus-export
00030         void SafeHaltCpu(unsigned cpu_idx);
00031     private:
00032         int interface_no;
00033
00034     };
00035
00036 }
00037
00038 #endif
```

8.114 LM32Cluster_Service.hpp

```
00001 #ifndef LM32Cluster_SERVICE_HPP_
00002 #define LM32Cluster_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class LM32Cluster;
00012     class LM32Cluster_Service : public saftbus::Service {
00013     public:
00014         LM32Cluster* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef LM32Cluster DriverType;
00017         LM32Cluster_Service(LM32Cluster* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         LM32Cluster_Service();
00019         ~LM32Cluster_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.115 Mailbox.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_MAILBOX_HPP_
00023 #define saftlib_MAILBOX_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include <MsiDevice.hpp>
00033
00034 #include <memory>
00035
00036 namespace saftlib {
00037
00038 class Mailbox : public MsiDevice {
00039     etherbone::Device &device;
00040     eb_address_t mailbox;
00041     eb_address_t mailbox_msi_first;
00042     friend class Slot;
00043
00048     void FreeSlot(int slot_index);
00049
00050 public:
00051     class Slot {
00052         Mailbox *mb;
00053         int slot_index;
00054         Slot(Mailbox *mailbox, int index);
00055         friend class Mailbox;
00056     public:
00058         ~Slot();
00059
00062         int getIndex();
00063
00066         eb_address_t getAddress();
00067
00072         void Use(uint32_t value);
00073     };
00074
00079     void UseSlot(int slot_index, uint32_t value);
00080
00081     Mailbox(etherbone::Device &device);
00086     std::unique_ptr<Slot> ConfigureSlot(uint32_t target_address);
00087 };
00088
00089 }
00090
00091 #endif
```

8.116 MasterFunctionGenerator.hpp

```
00001 /*  Copyright (C) 2011-2016 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *
00005  ********************************************************************************
00006  *  This library is free software; you can redistribute it and/or
00007  *  modify it under the terms of the GNU Lesser General Public
00008  *  License as published by the Free Software Foundation; either
```

```
00009  *  version 3 of the License, or (at your option) any later version.
00010  *
00011  *  This library is distributed in the hope that it will be useful,
00012  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  *  Lesser General Public License for more details.
00015  *
00016  *  You should have received a copy of the GNU Lesser General Public
00017  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00018  *****************************************************************************
00019  */
00020 #ifndef MASTER_FUNCTION_GENERATOR_HPP_
00021 #define MASTER_FUNCTION_GENERATOR_HPP_
00022
00023 #include <deque>
00024
00025 //#include "interfaces/MasterFunctionGenerator.h"
00026 #include "Owned.hpp"
00027 #include "FunctionGeneratorImpl.hpp"
00028
00029 // @saftbus-export
00030 #include "Time.hpp"
00031 // @saftbus-export
00032 #include <string>
00033
00034 #include <boost/interprocess/managed_shared_memory.hpp>
00035 #include <boost/interprocess/allocators/allocator.hpp>
00036 #include <boost/interprocess/containers/vector.hpp>
00037 #include <boost/interprocess/containers/map.hpp>
00038
00039
00040 namespace saftlib {
00041
00042 class TimingReceiver;
00043
00044 typedef boost::interprocess::allocator<ParameterTuple,
      boost::interprocess::managed_shared_memory::segment_manager>  ShmemAllocator;
00045 typedef boost::interprocess::vector<ParameterTuple, ShmemAllocator> ParameterVector;
00046
00047
00048
00054 class MasterFunctionGenerator : public Owned
00055 {
00056   public:
00057
00058     MasterFunctionGenerator(saftbus::Container *container,
00060                             const std::string &object_path,
00061                             std::vector<std::shared_ptr<FunctionGeneratorImpl> > functionGenerators);
00062     ~MasterFunctionGenerator();
00063
00064
00065     // typedef MasterFunctionGenerator_Service ServiceType;
00066     // struct ConstructorType {
00067     //   std::string objectPath;
00068     //   TimingReceiver *tr;
00069     //   std::vector<std::shared_ptr<FunctionGeneratorImpl» functionGenerators;
00070     // };
00071
00072     // static std::shared_ptr<MasterFunctionGenerator> create(const ConstructorType& args);
00073
00074     // iMasterFunctionGenerator overrides
00075
00084     // @saftbus-export
00085     void Arm();
00086
00102     // @saftbus-export
00103     void Abort(bool wait_for_abort_ack);
00104
00111     // @saftbus-export
00112     void InitializeSharedMemory(const std::string& shared_memory_name);
00113
00114     // @saftbus-export
00115     void AppendParameterTuplesForBeamProcess(int beam_process, bool arm, bool wait_for_arm_ack);
00116
00155     // @saftbus-export
00156     bool AppendParameterSets(const std::vector< std::vector< int16_t > >& coeff_a, const std::vector<
      std::vector< int16_t > >& coeff_b, const std::vector< std::vector< int32_t > >& coeff_c, const
      std::vector< std::vector< unsigned char > >& step, const std::vector< std::vector< unsigned char > >&
      freq, const std::vector< std::vector< unsigned char > >& shift_a, const std::vector< std::vector<
      unsigned char > >& shift_b, bool arm, bool wait_for_arm_ack);
00157
00166     // @saftbus-export
00167     std::vector<uint32_t> ReadExecutedParameterCounts();
00168
00178     // @saftbus-export
00179     std::vector<uint64_t> ReadFillLevels();
00180
```

```
00186     // @saftbus-export
00187     void Flush();
00188
00196     // @saftbus-export
00197     void setStartTag(uint32_t val);
00198     // @saftbus-export
00199     uint32_t getStartTag() const;
00200
00207     // @saftbus-export
00208     void setGenerateIndividualSignals(bool newvalue);
00209     // @saftbus-export
00210     bool getGenerateIndividualSignals() const;
00211
00216     // @saftbus-export
00217     std::vector<std::string> ReadAllNames();
00218
00222     // @saftbus-export
00223     std::vector<std::string> ReadNames();
00224
00229     // @saftbus-export
00230     std::vector<int> ReadArmed();
00231
00235     // @saftbus-export
00236     std::vector<int> ReadEnabled();
00237
00241     // @saftbus-export
00242     std::vector<int> ReadRunning();
00243
00246     // @saftbus-export
00247     void ResetActiveFunctionGenerators();
00248
00256     // @saftbus-export
00257     void SetActiveFunctionGenerators(const std::vector<std::string> &names);
00258
00259
00260     // Signals
00261
00288     // @saftbus-export
00289     sigc::signal< void , std::string  , saftlib::Time , bool , bool , bool > SigStopped;
00290
00292     // @saftbus-export
00293     sigc::signal< void , std::string , bool  > Armed;
00294
00296     // @saftbus-export
00297     sigc::signal< void , std::string , bool  > Enabled;
00298
00300     // @saftbus-export
00301     sigc::signal< void , std::string , bool  > Running;
00302
00304     // @saftbus-export
00305     sigc::signal< void , std::string , saftlib::Time > SigStarted;
00306
00308     // @saftbus-export
00309     sigc::signal< void , std::string > Refill;
00310
00318     // @saftbus-export
00319     sigc::signal< void , saftlib::Time > SigAllStopped;
00320
00326     // @saftbus-export
00327     sigc::signal< void > AllArmed;
00328
00329
00330     std::string getObjectPath();
00331
00332   protected:
00333     void arm_all();
00334     void reset_all();
00335
00336     void ownerQuit();
00337
00338     void on_fg_running(std::shared_ptr<FunctionGeneratorImpl>& fg, bool);
00339     void on_fg_armed(std::shared_ptr<FunctionGeneratorImpl>& fg, bool);
00340     void on_fg_enabled(std::shared_ptr<FunctionGeneratorImpl>& fg, bool);
00341     void on_fg_started(std::shared_ptr<FunctionGeneratorImpl>& fg, uint64_t);
00342     void on_fg_stopped(std::shared_ptr<FunctionGeneratorImpl>& fg, uint64_t time, bool abort, bool
      hardwareUnderflow, bool microcontrollerUnderflow);
00343     void on_fg_refill(std::shared_ptr<FunctionGeneratorImpl>& fg);
00344
00345     bool all_armed();
00346     bool all_stopped();
00347     bool WaitTimeout();
00348     void waitForCondition(std::function<bool()> condition, int timeout_ms);
00349
00350     //TimingReceiver *tr;
00351     std::string objectPath;
00352     std::vector<std::shared_ptr<FunctionGeneratorImpl» allFunctionGenerators;
00353     std::vector<std::shared_ptr<FunctionGeneratorImpl» activeFunctionGenerators;
```

```
00354     uint32_t startTag;
00355     bool generateIndividualSignals;
00356     sigc::connection waitTimeout;
00357 
00358     std::map <int,std::vector<ParameterTuple» parametersForBeamProcess;
00359     std::unique_ptr<boost::interprocess::managed_shared_memory> shm_params;
00360     boost::interprocess::interprocess_mutex* shm_mutex;
00361     std::map<std::string,ParameterVector*> paramVectors;
00362 };
00363 
00364 }
00365 
00366 #endif
```

8.117 MasterFunctionGenerator_Proxy.hpp

```
00001 #ifndef MasterFunctionGenerator_PROXY_HPP_
00002 #define MasterFunctionGenerator_PROXY_HPP_
00003 
00004 #include <saftbus/client.hpp>
00005 
00006 #include <sigc++/sigc++.h>
00007 
00008 #include "Time.hpp"
00009 #include <string>
00010 
00011 #include "Owned_Proxy.hpp"
00012 
00013 namespace saftlib {
00014 
00020     class MasterFunctionGenerator_Proxy : public  Owned_Proxy   {
00021         static std::vector<std::string> gen_interface_names();
00022     public:
00023         MasterFunctionGenerator_Proxy(const std::string &object_path, saftbus::SignalGroup
      &signal_group = saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00024         static std::shared_ptr<MasterFunctionGenerator_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00025         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00034     // @saftbus-export
00035         void Arm();
00051     // @saftbus-export
00052         void Abort(bool wait_for_abort_ack);
00059     // @saftbus-export
00060         void InitializeSharedMemory(const std::string& shared_memory_name);
00061     // @saftbus-export
00062         void AppendParameterTuplesForBeamProcess(int beam_process, bool arm, bool wait_for_arm_ack);
00101     // @saftbus-export
00102         bool AppendParameterSets(const std::vector< std::vector< int16_t > >& coeff_a, const
      std::vector< std::vector< int16_t > >& coeff_b, const std::vector< std::vector< int32_t > >& coeff_c,
      const std::vector< std::vector< unsigned char > >& step, const std::vector< std::vector< unsigned char
      > >& freq, const std::vector< std::vector< unsigned char > >& shift_a, const std::vector< std::vector<
      unsigned char > >& shift_b, bool arm, bool wait_for_arm_ack);
00111     // @saftbus-export
00112         std::vector<uint32_t> ReadExecutedParameterCounts();
00122     // @saftbus-export
00123         std::vector<uint64_t> ReadFillLevels();
00129     // @saftbus-export
00130         void Flush();
00138     // @saftbus-export
00139         void setStartTag(uint32_t val);
00140     // @saftbus-export
00141         uint32_t getStartTag();
00148     // @saftbus-export
00149         void setGenerateIndividualSignals(bool newvalue);
00150     // @saftbus-export
00151         bool getGenerateIndividualSignals();
00156     // @saftbus-export
00157         std::vector<std::string> ReadAllNames();
00161     // @saftbus-export
00162         std::vector<std::string> ReadNames();
00167     // @saftbus-export
00168         std::vector<int> ReadArmed();
00172     // @saftbus-export
00173         std::vector<int> ReadEnabled();
00177     // @saftbus-export
00178         std::vector<int> ReadRunning();
00181     // @saftbus-export
00182         void ResetActiveFunctionGenerators();
00190     // @saftbus-export
00191         void SetActiveFunctionGenerators(const std::vector<std::string> & names);
00218     // @saftbus-export
00219         sigc::signal<void, std::string, saftlib::Time, bool, bool, bool> SigStopped;
```

```
00221     // @saftbus-export
00222         sigc::signal<void, std::string, bool> Armed;
00224     // @saftbus-export
00225         sigc::signal<void, std::string, bool> Enabled;
00227     // @saftbus-export
00228         sigc::signal<void, std::string, bool> Running;
00230     // @saftbus-export
00231         sigc::signal<void, std::string, saftlib::Time> SigStarted;
00233     // @saftbus-export
00234         sigc::signal<void, std::string> Refill;
00242     // @saftbus-export
00243         sigc::signal<void, saftlib::Time> SigAllStopped;
00249     // @saftbus-export
00250         sigc::signal<void> AllArmed;
00251     private:
00252         int interface_no;
00253
00254     };
00255
00256 }
00257
00258 #endif
```

8.118 MasterFunctionGenerator_Service.hpp

```
00001 #ifndef MasterFunctionGenerator_SERVICE_HPP_
00002 #define MasterFunctionGenerator_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class MasterFunctionGenerator;
00012     class MasterFunctionGenerator_Service : public saftbus::Service {
00013     public:
00014         MasterFunctionGenerator* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef MasterFunctionGenerator DriverType;
00017         MasterFunctionGenerator_Service(MasterFunctionGenerator* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         MasterFunctionGenerator_Service();
00019         ~MasterFunctionGenerator_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void SigStopped_dispatch_function(std::string arg_1,saftlib::Time arg_2,bool arg_3,bool
      arg_4,bool arg_5);
00023         sigc::connection SigStopped_connection;
00024         void Armed_dispatch_function(std::string arg_1,bool arg_2);
00025         sigc::connection Armed_connection;
00026         void Enabled_dispatch_function(std::string arg_1,bool arg_2);
00027         sigc::connection Enabled_connection;
00028         void Running_dispatch_function(std::string arg_1,bool arg_2);
00029         sigc::connection Running_connection;
00030         void SigStarted_dispatch_function(std::string arg_1,saftlib::Time arg_2);
00031         sigc::connection SigStarted_connection;
00032         void Refill_dispatch_function(std::string arg_1);
00033         sigc::connection Refill_connection;
00034         void SigAllStopped_dispatch_function(saftlib::Time arg_1);
00035         sigc::connection SigAllStopped_connection;
00036         void AllArmed_dispatch_function();
00037         sigc::connection AllArmed_connection;
00038         void Destroyed_dispatch_function();
00039         sigc::connection Destroyed_connection;
00040     };
00041
00042 }
00043
00044 #endif
```

8.119 MsiDevice.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
```

```
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef saftlib_MSI_DEVICE_HPP_
00023 #define saftlib_MSI_DEVICE_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include <SdbDevice.hpp>
00033
00034 namespace saftlib {
00035
00036 class MsiDevice : public SdbDevice {
00037     friend class SAFTd; // SAFTd can use an MsiDevice to register a callbak on MSIs
00038 protected:
00039     etherbone::sdb_msi_device msi_device;
00040 public:
00041     MsiDevice(etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID);
00042
00043 };
00044
00045 }
00046
00047 #endif
```

8.120 OpenDevice.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef saftlib_OPEN_DEVICE_HPP_
00023 #define saftlib_OPEN_DEVICE_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include <saftbus/loop.hpp>
00033
00034 #include <memory>
00035
00036 #include <sys/stat.h>
00037
```

```
00038 namespace saftlib {
00039
00040 class SAFTd;
00041 class IRQ;
00042 class Mailbox;
00043 class EB_Forward;
00056 class OpenDevice {
00057 protected:
00058     std::string etherbone_path;
00059     struct stat dev_stat;
00060     etherbone::Device device;
00061
00062 public:
00068     OpenDevice(const etherbone::Socket &socket, const std::string& etherbone_path, int
      polling_interval_ms = 1, SAFTd *saftd = nullptr);
00069     virtual ~OpenDevice();
00070
00071     etherbone::Device &get_device();
00072
00076     // @saftbus-export
00077     std::string getEtherbonePath() const;
00078
00079
00086     // @saftbus-export
00087     std::string getEbForwardPath() const;
00088
00089 private:
00090     // etherbone forwading
00091     std::unique_ptr<EB_Forward> eb_forward;
00092     std::string eb_forward_path;
00093
00094     // polling for MSIs on hardware that doesn't support real MSIs
00095     bool poll_msi(bool only_once);
00096     int polling_interval_ms;
00097     saftbus::SourceHandle poll_timeout_source;
00098     saftbus::SourceHandle poll_once;
00099
00100     // following members are for testing MSI capability (real or polled MSIs)
00101     void check_msi_callback(eb_data_t value);
00102     std::unique_ptr<IRQ> check_irq; // IRQ to check MSI capability
00103     SAFTd *saftd;         // a pointer to SAFTd is needed because in case of polled MSIs the
      OpenDevice needs to call SAFTds write function
00104     eb_address_t first, last, mask; // range of addresses that are valid for MSI
00105     eb_address_t msi_first, msi_last;         // address offset which needs to be subtracted
00106     eb_address_t irq_adr; // the MSI callback function is registered under this address, and the
      Mailbox is configured with irq_adr+msi_first
00107     bool check_msi_phase, needs_polling; // check_msi_phase is true until the MSI type was determined.
00108                                          // needs_polling is false in the beginning. It is set to true
      if the poll_msi function receives the injected MSI.
00109 };
00110
00111
00112 } // namespace
00113
00114 #endif
```

8.121 OpenDevice_Proxy.hpp

```
00001 #ifndef OpenDevice_PROXY_HPP_
00002 #define OpenDevice_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008
00009 namespace saftlib {
00010
00023     class OpenDevice_Proxy : public virtual saftbus::Proxy
00024     {
00025         static std::vector<std::string> gen_interface_names();
00026     public:
00027         OpenDevice_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00028         static std::shared_ptr<OpenDevice_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00029         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00033     // @saftbus-export
00034         std::string getEtherbonePath();
00041     // @saftbus-export
00042         std::string getEbForwardPath();
00043     private:
```

```
00044         int interface_no;
00045
00046     };
00047
00048 }
00049
00050 #endif
```

8.122 OpenDevice_Service.hpp

```
00001 #ifndef OpenDevice_SERVICE_HPP_
00002 #define OpenDevice_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class OpenDevice;
00012     class OpenDevice_Service : public saftbus::Service {
00013     public:
00014         OpenDevice* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef OpenDevice DriverType;
00017         OpenDevice_Service(OpenDevice* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         OpenDevice_Service();
00019         ~OpenDevice_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.123 Output.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2023 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_OUTPUT_HPP
00023 #define saftlib_OUTPUT_HPP
00024
00025 #include "Io.hpp"
00026 #include "ActionSink.hpp"
00027
00028 namespace saftlib {
00029
00040 class Output : public ActionSink
00041 {
00042     public:
00043         Output(ECA &eca
00044               , Io &io
00045               , const std::string &output_object_path
00046               , const std::string &input_partner_path
00047               , unsigned channel
00048               , saftbus::Container *container = nullptr);
```

```
00049
00065         // @saftbus-export
00066         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, bool on);
00067
00068
00077         // @saftbus-export
00078         void WriteOutput(bool value);
00079
00080
00087         // @saftbus-export
00088         bool ReadOutput();
00089
00096         // @saftbus-export
00097         bool ReadCombinedOutput();
00098
00099
00105         // @saftbus-export
00106         bool StartClock(double high_phase, double low_phase, uint64_t phase_offset);
00107
00108
00114         // @saftbus-export
00115         bool ClockStatus(double &high_phase, double &low_phase, uint64_t &phase_offset);
00116
00120         // @saftbus-export
00121         bool StopClock();
00122
00126         // @saftbus-export
00127         uint32_t getIndexOut() const;
00128
00135         // @saftbus-export
00136         void setOutputEnable(bool val);
00137         // @saftbus-export
00138         bool getOutputEnable() const;
00139
00140
00144         // @saftbus-export
00145         void setSpecialPurposeOut(bool val);
00146         // @saftbus-export
00147         bool getSpecialPurposeOut() const;
00148
00152         // @saftbus-export
00153         void setGateOut(bool val);
00154         // @saftbus-export
00155         bool getGateOut() const;
00156
00160         // @saftbus-export
00161         void setBuTiSMultiplexer(bool val);
00162         // @saftbus-export
00163         bool getBuTiSMultiplexer() const;
00164
00165
00169         // @saftbus-export
00170         void setPPSMultiplexer(bool val);
00171         // @saftbus-export
00172         bool getPPSMultiplexer() const;
00173
00177         // @saftbus-export
00178         bool getOutputEnableAvailable() const;
00179
00183         // @saftbus-export
00184         bool getSpecialPurposeOutAvailable() const;
00185
00189         // @saftbus-export
00190         std::string getLogicLevelOut() const;
00191
00195         // @saftbus-export
00196         std::string getTypeOut() const;
00197
00201         // @saftbus-export
00202         std::string getInput() const;
00203
00204
00205
00206         // Property signals
00207         //   sigc::signal< void, bool > OutputEnable;
00208         //   sigc::signal< void, bool > SpecialPurposeOut;
00209         //   sigc::signal< void, bool > BuTiSMultiplexer;
00210
00211     protected:
00212         Io &io;
00213         std::string partnerPath;
00214
00215         double   clk_low_phase;
00216         double   clk_high_phase;
00217         uint64_t clk_phase_offset;
00218 };
00219
```

```
00220 }
00221
00222 #endif
```

8.124 Output_Proxy.hpp

```
00001 #ifndef Output_PROXY_HPP_
00002 #define Output_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "ActionSink_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00022     class Output_Proxy : public  ActionSink_Proxy   {
00023         static std::vector<std::string> gen_interface_names();
00024     public:
00025         Output_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00026         static std::shared_ptr<Output_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00027         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00043         // @saftbus-export
00044         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, bool on);
00053         // @saftbus-export
00054         void WriteOutput(bool value);
00061         // @saftbus-export
00062         bool ReadOutput();
00069         // @saftbus-export
00070         bool ReadCombinedOutput();
00076         // @saftbus-export
00077         bool StartClock(double high_phase, double low_phase, uint64_t phase_offset);
00083         // @saftbus-export
00084         bool ClockStatus(double & high_phase, double & low_phase, uint64_t & phase_offset);
00088         // @saftbus-export
00089         bool StopClock();
00093         // @saftbus-export
00094         uint32_t getIndexOut();
00101         // @saftbus-export
00102         void setOutputEnable(bool val);
00103         // @saftbus-export
00104         bool getOutputEnable();
00108         // @saftbus-export
00109         void setSpecialPurposeOut(bool val);
00110         // @saftbus-export
00111         bool getSpecialPurposeOut();
00115         // @saftbus-export
00116         void setGateOut(bool val);
00117         // @saftbus-export
00118         bool getGateOut();
00122         // @saftbus-export
00123         void setBuTiSMultiplexer(bool val);
00124         // @saftbus-export
00125         bool getBuTiSMultiplexer();
00129         // @saftbus-export
00130         void setPPSMultiplexer(bool val);
00131         // @saftbus-export
00132         bool getPPSMultiplexer();
00136         // @saftbus-export
00137         bool getOutputEnableAvailable();
00141         // @saftbus-export
00142         bool getSpecialPurposeOutAvailable();
00146         // @saftbus-export
00147         std::string getLogicLevelOut();
00151         // @saftbus-export
00152         std::string getTypeOut();
00156         // @saftbus-export
00157         std::string getInput();
00158     private:
00159         int interface_no;
00160
00161     };
00162
00163 }
00164
00165 #endif
```

8.125 Output_Service.hpp

```
00001 #ifndef Output_SERVICE_HPP_
00002 #define Output_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class Output;
00012     class Output_Service : public saftbus::Service {
00013     public:
00014         Output* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef Output DriverType;
00017         Output_Service(Output* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         Output_Service();
00019         ~Output_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void OverflowCount_dispatch_function(uint64_t arg_1);
00023         sigc::connection OverflowCount_connection;
00024         void ActionCount_dispatch_function(uint64_t arg_1);
00025         sigc::connection ActionCount_connection;
00026         void LateCount_dispatch_function(uint64_t arg_1);
00027         sigc::connection LateCount_connection;
00028         void SigLate_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00029         sigc::connection SigLate_connection;
00030         void EarlyCount_dispatch_function(uint64_t arg_1);
00031         sigc::connection EarlyCount_connection;
00032         void SigEarly_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00033         sigc::connection SigEarly_connection;
00034         void ConflictCount_dispatch_function(uint64_t arg_1);
00035         sigc::connection ConflictCount_connection;
00036         void SigConflict_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00037         sigc::connection SigConflict_connection;
00038         void DelayedCount_dispatch_function(uint64_t arg_1);
00039         sigc::connection DelayedCount_connection;
00040         void SigDelayed_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00041         sigc::connection SigDelayed_connection;
00042         void Destroyed_dispatch_function();
00043         sigc::connection Destroyed_connection;
00044     };
00045
00046 }
00047
00048 #endif
```

8.126 OutputCondition.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_OUTPUT_CONDITION_HPP
00023 #define saftlib_OUTPUT_CONDITION_HPP
00024
```

```
00025 #include "Owned.hpp"
00026 #include "Condition.hpp"
00027
00028 // @saftbus-include
00029 #include <Time.hpp>
00030
00031 #include <saftbus/service.hpp>
00032
00033 #include <functional>
00034
00035 namespace saftlib {
00036
00037 class OutputCondition_Service;
00038
00039 class OutputCondition : public Condition
00040 {
00041 public:
00042     OutputCondition(ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask,
      int64_t offset, uint32_t tag, saftbus::Container *container);
00043
00050     // @saftbus-export
00051     bool getOn() const;
00052     // @saftbus-export
00053     void setOn(bool val);
00054
00055     // this typedef is needed for the ActionSink::NewCondition template function
00056     typedef OutputCondition_Service ServiceType;
00057 };
00058
00059 }
00060
00061 #endif
```

8.127 OutputCondition_Proxy.hpp

```
00001 #ifndef OutputCondition_PROXY_HPP_
00002 #define OutputCondition_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007 #include <Time.hpp>
00008
00009 #include "Condition_Proxy.hpp"
00010
00011 namespace saftlib {
00012
00013 // @saftbus-include
00014     class OutputCondition_Proxy : public  Condition_Proxy   {
00015         static std::vector<std::string> gen_interface_names();
00016     public:
00017         OutputCondition_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00018         static std::shared_ptr<OutputCondition_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00019         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00026     // @saftbus-export
00027         bool getOn();
00028     // @saftbus-export
00029         void setOn(bool val);
00030     private:
00031         int interface_no;
00032
00033     };
00034
00035 }
00036
00037 #endif
```

8.128 OutputCondition_Service.hpp

```
00001 #ifndef OutputCondition_SERVICE_HPP_
00002 #define OutputCondition_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
```

```
00008
00009 namespace saftlib {
00010
00011     class OutputCondition;
00012     class OutputCondition_Service : public saftbus::Service {
00013     public:
00014         OutputCondition* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef OutputCondition DriverType;
00017         OutputCondition_Service(OutputCondition* instance, std::function<void()> destruction_callback
      = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         OutputCondition_Service();
00019         ~OutputCondition_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.129 Owned.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_OWNED_HPP_
00023 #define saftlib_OWNED_HPP_
00024
00025 #include <saftbus/service.hpp>
00026
00027 #include <sigc++/sigc++.h>
00028
00029 namespace saftlib {
00038     class Owned
00039     {
00040     public:
00041         Owned(saftbus::Container *container);
00042         virtual ~Owned();
00043
00048         void set_service(saftbus::Service *service);
00049
00052         void release_service();
00053
00060         // @saftbus-export
00061         void Disown();
00062
00063
00068         // @saftbus-export
00069         void Own();
00070
00079         // @saftbus-export
00080         std::string getOwner() const;
00081
00082
00089         // @saftbus-export
00090         bool getDestructible() const;
00091
00092
00098         // @saftbus-export
00099         void Destroy();
00100
```

```
00109         // @saftbus-signal
00110         sigc::signal<void> Destroyed;
00111
00112     protected:
00114         void ownerOnly() const;
00115     private:
00116         saftbus::Container *cont;
00117         saftbus::Service *service;
00118
00119     };
00120
00121 }
00122
00123 #endif
```

8.130 Owned_Proxy.hpp

```
00001 #ifndef Owned_PROXY_HPP_
00002 #define Owned_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006 #include <sigc++/sigc++.h>
00007
00008
00009
00010 namespace saftlib {
00011
00020     class Owned_Proxy : public virtual saftbus::Proxy
00021     {
00022         static std::vector<std::string> gen_interface_names();
00023     public:
00024         Owned_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00025         static std::shared_ptr<Owned_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00026         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00033         // @saftbus-export
00034         void Disown();
00039         // @saftbus-export
00040         void Own();
00049         // @saftbus-export
00050         std::string getOwner();
00057         // @saftbus-export
00058         bool getDestructible();
00064         // @saftbus-export
00065         void Destroy();
00074         // @saftbus-signal
00075         sigc::signal<void> Destroyed;
00076     private:
00077         int interface_no;
00078
00079     };
00080
00081 }
00082
00083 #endif
```

8.131 Owned_Service.hpp

```
00001 #ifndef Owned_SERVICE_HPP_
00002 #define Owned_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class Owned;
00012     class Owned_Service : public saftbus::Service {
00013     public:
00014         Owned* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef Owned DriverType;
00017         Owned_Service(Owned* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
```

```
00018         Owned_Service();
00019         ~Owned_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.132 Reset.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef SAFTLIB_RESET_HPP_
00023 #define SAFTLIB_RESET_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "SdbDevice.hpp"
00033
00034 namespace saftlib {
00035
00036 class Reset : public SdbDevice {
00037 public:
00038     Reset(etherbone::Device &device);
00043     // @saftbus-export
00044     void WdRetrigger();
00045
00049     // @saftbus-export
00050     void CpuHalt(unsigned idx);
00051
00055     // @saftbus-export
00056     void CpuReset(unsigned idx);
00057
00061     // @saftbus-export
00062     uint32_t CpuHaltStatus();
00063 };
00064
00065 }
00066
00067 #endif
```

8.133 Reset_Proxy.hpp

```
00001 #ifndef Reset_PROXY_HPP_
00002 #define Reset_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
```

```
00008
00009 namespace saftlib {
00010
00011     class Reset_Proxy : public virtual saftbus::Proxy
00012     {
00013         static std::vector<std::string> gen_interface_names();
00014     public:
00015         Reset_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00016         static std::shared_ptr<Reset_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00017         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00022     // @saftbus-export
00023         void WdRetrigger();
00027     // @saftbus-export
00028         void CpuHalt(unsigned idx);
00032     // @saftbus-export
00033         void CpuReset(unsigned idx);
00037     // @saftbus-export
00038         uint32_t CpuHaltStatus();
00039     private:
00040         int interface_no;
00041
00042     };
00043
00044 }
00045
00046 #endif
```

8.134 Reset_Service.hpp

```
00001 #ifndef Reset_SERVICE_HPP_
00002 #define Reset_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class Reset;
00012     class Reset_Service : public saftbus::Service {
00013     public:
00014         Reset* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef Reset DriverType;
00017         Reset_Service(Reset* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         Reset_Service();
00019         ~Reset_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.135 SAFTd.hpp

```
00001 /* *  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
```

```
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef TR_SAFTD_HPP_
00023 #define TR_SAFTD_HPP_
00024
00025 #include <saftbus/service.hpp>
00026
00027 #include <memory>
00028 #include <string>
00029 #include <functional>
00030 #include <map>
00031
00032 #include "TimingReceiver.hpp"
00033 #include "eb-forward.hpp"
00034 #include "MsiDevice.hpp"
00035
00046 namespace saftlib {
00047
00062     class SAFTd : public etherbone::Handler {
00063         // @saftbus-default-object-path /de/gsi/saftlib
00064     public:
00067         SAFTd(saftbus::Container *container = nullptr);
00068         ~SAFTd();
00069
00089         // @saftbus-export
00090         std::string AttachDevice(const std::string& name, const std::string& path, int
      polling_interval_ms = 1);
00091
00096         // @saftbus-export
00097         void RemoveDevice(const std::string& name);
00098
00104         // @saftbus-export
00105         void Quit();
00106
00112         // @saftbus-export
00113         std::string getSourceVersion() const;
00114
00121         // @saftbus-export
00122         std::string getBuildInfo() const;
00123
00124
00134         // @saftbus-export
00135         std::map< std::string, std::string > getDevices() const;
00136
00142         // @saftbus-export
00143         std::string EbForward(const std::string& saftlib_device);
00144
00147         void release_irq(eb_address_t irq);
00148
00149
00150
00159         std::unique_ptr<IRQ> request_irq(MsiDevice &msi, const std::function<void(eb_data_t)>& slot);
00160
00163         std::string getObjectPath();
00164
00166         etherbone::Socket &get_etherbone_socket() { return socket; }
00167
00170         TimingReceiver* getTimingReceiver(const std::string &object_path);
00171
00177         eb_status_t read (eb_address_t address, eb_width_t width, eb_data_t* data);
00179         eb_status_t write(eb_address_t address, eb_width_t width, eb_data_t data);
00180
00181     private:
00182
00183
00190         bool request_irq(eb_address_t irq, const std::function<void(eb_data_t)>& slot);
00191
00192         void RemoveObject(const std::string& name);
00193
00194         // The sdb structure for this "virtual" etherbone device
00195         sdb_device eb_slave_sdb;
00196
00197
00198         // Need a pointer to saftbus::Container to insert new Service Objects (instances of
      TimingReceiver_Service)
00199         // and the object_path of the SAFTd_Service (normally "/de/gsi/saftlib") as prefix for the
      object_path of TimingReceiver_Service objects
00200         saftbus::Container *container;
00201         std::string object_path;
00202
00203         //
00204         etherbone::Socket socket;
00205         saftbus::SourceHandle eb_source;
00206
```

```
00207         std::map< std::string, std::unique_ptr<EB_Forward> > eb_forward;
00208
00209         // remember all attached devices
00210         std::map<std::string, std::unique_ptr<TimingReceiver> > attached_devices;
00211
00212         std::map<eb_address_t, std::function<void(eb_data_t)> > irqs;
00213
00214         bool quit;
00215
00216     };
00217
00225     class IRQ {
00226         friend class SAFTd;
00227         IRQ(SAFTd *sd, eb_address_t a, eb_address_t f);
00228         SAFTd *saftd;
00229         eb_address_t addr;
00230         eb_address_t msi_device_first;
00231     public:
00232         ~IRQ();
00234         eb_address_t address();
00235     };
00236
00237
00238 }
00239
00240 #endif
00241
```

8.136 SAFTd_Proxy.hpp

```
00001 #ifndef SAFTd_PROXY_HPP_
00002 #define SAFTd_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008
00009 namespace saftlib {
00010
00025     class SAFTd_Proxy : public virtual saftbus::Proxy
00026     {
00027         static std::vector<std::string> gen_interface_names();
00028     public:
00029         SAFTd_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00030         static std::shared_ptr<SAFTd_Proxy> create(const std::string &object_path = "/de/gsi/saftlib",
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00031         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00051         // @saftbus-export
00052         std::string AttachDevice(const std::string& name, const std::string& path, int
      polling_interval_ms = 1);
00057         // @saftbus-export
00058         void RemoveDevice(const std::string& name);
00064         // @saftbus-export
00065         void Quit();
00071         // @saftbus-export
00072         std::string getSourceVersion();
00079         // @saftbus-export
00080         std::string getBuildInfo();
00090         // @saftbus-export
00091         std::map< std::string, std::string > getDevices();
00097         // @saftbus-export
00098         std::string EbForward(const std::string& saftlib_device);
00099     private:
00100         int interface_no;
00101
00102     };
00103
00104 }
00105
00106 #endif
```

8.137 SAFTd_Service.hpp

```
00001 #ifndef SAFTd_SERVICE_HPP_
00002 #define SAFTd_SERVICE_HPP_
00003
```

```
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class SAFTd;
00012     class SAFTd_Service : public saftbus::Service {
00013     public:
00014         SAFTd* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef SAFTd DriverType;
00017         SAFTd_Service(SAFTd* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         SAFTd_Service();
00019         ~SAFTd_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.138 SCUbusActionSink.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef SAFTLIB_SCUBUS_ACTION_SINK_HPP_
00023 #define SAFTLIB_SCUBUS_ACTION_SINK_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "ActionSink.hpp"
00033
00034 namespace saftlib {
00035
00045 class SCUbusActionSink : public ActionSink {
00046     public:
00047         SCUbusActionSink(etherbone::Device &device
00048                          , ECA &eca
00049                          , const std::string &object_path
00050                          , const std::string &name
00051                          , unsigned channel, eb_address_t scubus_address
00052                          , saftbus::Container *container = nullptr);
00053
00069         // @saftbus-export
00070         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t
      tag);
00071
00072
00079         // @saftbus-export
00080         void InjectTag(uint32_t tag);
00081
00082     protected:
00083         etherbone::Device &device;
00084         eb_address_t scubus;
```

```
00085 };
00086
00087 }
00088
00089 #endif
```

8.139 SCUbusActionSink_Proxy.hpp

```
00001 #ifndef SCUbusActionSink_PROXY_HPP_
00002 #define SCUbusActionSink_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "ActionSink_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00021     class SCUbusActionSink_Proxy : public  ActionSink_Proxy   {
00022         static std::vector<std::string> gen_interface_names();
00023     public:
00024         SCUbusActionSink_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00025         static std::shared_ptr<SCUbusActionSink_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00026         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00042         // @saftbus-export
00043         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t
      tag);
00050         // @saftbus-export
00051         void InjectTag(uint32_t tag);
00052     private:
00053         int interface_no;
00054
00055     };
00056
00057 }
00058
00059 #endif
```

8.140 SCUbusActionSink_Service.hpp

```
00001 #ifndef SCUbusActionSink_SERVICE_HPP_
00002 #define SCUbusActionSink_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class SCUbusActionSink;
00012     class SCUbusActionSink_Service : public saftbus::Service {
00013     public:
00014         SCUbusActionSink* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef SCUbusActionSink DriverType;
00017         SCUbusActionSink_Service(SCUbusActionSink* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         SCUbusActionSink_Service();
00019         ~SCUbusActionSink_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void OverflowCount_dispatch_function(uint64_t arg_1);
00023         sigc::connection OverflowCount_connection;
00024         void ActionCount_dispatch_function(uint64_t arg_1);
00025         sigc::connection ActionCount_connection;
00026         void LateCount_dispatch_function(uint64_t arg_1);
00027         sigc::connection LateCount_connection;
00028         void SigLate_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00029         sigc::connection SigLate_connection;
00030         void EarlyCount_dispatch_function(uint64_t arg_1);
00031         sigc::connection EarlyCount_connection;
```

```
00032         void SigEarly_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00033         sigc::connection SigEarly_connection;
00034         void ConflictCount_dispatch_function(uint64_t arg_1);
00035         sigc::connection ConflictCount_connection;
00036         void SigConflict_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00037         sigc::connection SigConflict_connection;
00038         void DelayedCount_dispatch_function(uint64_t arg_1);
00039         sigc::connection DelayedCount_connection;
00040         void SigDelayed_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00041         sigc::connection SigDelayed_connection;
00042         void Destroyed_dispatch_function();
00043         sigc::connection Destroyed_connection;
00044     };
00045
00046 }
00047
00048 #endif
```

8.141 SCUbusCondition.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef SCUBUS_CONDITION_HPP_
00023 #define SCUBUS_CONDITION_HPP_
00024
00025 #include <saftbus/service.hpp>
00026
00027 #include "Owned.hpp"
00028 #include "Condition.hpp"
00029 #include "ActionSink.hpp"
00030
00031 namespace saftlib {
00032
00033
00034 class SCUbusCondition_Service;
00035
00036
00043 class SCUbusCondition : public Condition {
00044 public:
00045     SCUbusCondition(ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask,
      int64_t offset, uint32_t tag, saftbus::Container *container = nullptr);
00046
00049     // @saftbus-export
00050     uint32_t getTag() const;
00053     // @saftbus-export
00054     void setTag(uint32_t val);
00055
00056     typedef SCUbusCondition_Service ServiceType;
00057 };
00058
00059 }
00060
00061 #endif
```

8.142 SCUbusCondition_Proxy.hpp

```
00001 #ifndef SCUbusCondition_PROXY_HPP_
```

```
00002 #define SCUbusCondition_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "Condition_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00018     class SCUbusCondition_Proxy : public  Condition_Proxy   {
00019         static std::vector<std::string> gen_interface_names();
00020     public:
00021         SCUbusCondition_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00022         static std::shared_ptr<SCUbusCondition_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00023         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00026     // @saftbus-export
00027         uint32_t getTag();
00030     // @saftbus-export
00031         void setTag(uint32_t val);
00032     private:
00033         int interface_no;
00034
00035     };
00036
00037 }
00038
00039 #endif
```

8.143 SCUbusCondition_Service.hpp

```
00001 #ifndef SCUbusCondition_SERVICE_HPP_
00002 #define SCUbusCondition_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class SCUbusCondition;
00012     class SCUbusCondition_Service : public saftbus::Service {
00013     public:
00014         SCUbusCondition* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef SCUbusCondition DriverType;
00017         SCUbusCondition_Service(SCUbusCondition* instance, std::function<void()> destruction_callback
      = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         SCUbusCondition_Service();
00019         ~SCUbusCondition_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.144 SdbDevice.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
```

```
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_SDB_DEVICE_HPP_
00023 #define saftlib_SDB_DEVICE_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 namespace saftlib {
00033
00046 class SdbDevice {
00047     friend class SAFTd; // SAFTd can use an MsiDevice to register a callbak on MSIs
00048 protected:
00049     eb_address_t adr_first;
00050     etherbone::Device &device;
00051 public:
00052     SdbDevice(etherbone::Device &device, uint32_t VENDOR_ID, uint32_t DEVICE_ID, bool
      throw_if_not_found = true);
00053     virtual ~SdbDevice();
00054 };
00055
00056 }
00057
00058 #endif
```

8.145 SerdesClockGen.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef SER_CLK_GEN_REGS_H
00023 #define SER_CLK_GEN_REGS_H
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "SdbDevice.hpp"
00033
00034 #include <stdint.h>
00035
00036 #define IO_SER_CLK_GEN_PRODUCT_ID    0x5f3eaf43
00037 #define IO_SER_CLK_GEN_VENDOR_ID     0x00000651
00038
00039 #define IO_SER_CLK_GEN_BITS          8
00040
00041 #define IO_SER_CLK_GEN_DEBUG_MODE    0
00042
00043 namespace saftlib {
00044
00045
```

```
00046 class SerdesClockGen : public SdbDevice
00047 {
00048
00049     typedef enum
00050     {
00051         eSCK_selr      = 0x00,
00052         eSCK_perr      = 0x04,
00053         eSCK_perhir    = 0x08,
00054         eSCK_fracr     = 0x0c,
00055         eSCK_normmaskr = 0x10,
00056         eSCK_skipmaskr = 0x14,
00057         eSCK_phofslr   = 0x18,
00058         eSCK_phofshr   = 0x1c,
00059     } e_SerClkGen_RegisterArea;
00060
00061     typedef struct SerClkGenControl
00062     {
00063         uint32_t period_integer;
00064         uint32_t period_high;
00065         uint32_t period_fraction;
00066         uint16_t bit_pattern_normal;
00067         uint16_t bit_pattern_skip;
00068         uint64_t phase_offset;
00069     } s_SerClkGenControl;
00070
00071     // eb_address_t clkgen_address;
00072     // etherbone::Device &device;
00073
00074     static void CalcClockParameters(double hi, double lo, uint64_t phase, struct SerClkGenControl
      *control);
00075 public:
00076     SerdesClockGen(etherbone::Device &device);
00077
00078     bool StartClock(int io_channel, int io_index, double high_phase, double low_phase, uint64_t
      phase_offset);
00079     bool StopClock(int io_channel, int io_index);
00080     bool ConfigureClock(int io_channel, int io_index, double high_phase, double low_phase, uint64_t
      phase_offset);
00081 };
00082
00083 } // namespace
00084
00085 #endif
```

8.146 SoftwareActionSink.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_SOFTWARE_ACTION_SINK_H
00023 #define saftlib_SOFTWARE_ACTION_SINK_H
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "ActionSink.hpp"
00033
00034 namespace saftlib {
00035
00036     class SoftwareCondition;
00037     class ECA;
```

```
00052     class SoftwareActionSink : public ActionSink {
00053     public:
00054         SoftwareActionSink(ECA &eca
00055                            , const std::string &object_path
00056                            , const std::string &name
00057                            , unsigned channel, unsigned num, eb_address_t queue_address
00058                            , saftbus::Container *container = nullptr);
00059 
00074         // @saftbus-export
00075         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset);
00076 
00077         // override receiveMSI to also pop the software queue
00078         void receiveMSI(uint8_t code);
00079 
00080         SoftwareCondition * getCondition(const std::string object_path);
00081 
00082     protected:
00083         eb_address_t queue;
00084     };
00085 
00086 }
00087 
00088 #endif
```

8.147 SoftwareActionSink_Proxy.hpp

```
00001 #ifndef SoftwareActionSink_PROXY_HPP_
00002 #define SoftwareActionSink_PROXY_HPP_
00003 
00004 #include <saftbus/client.hpp>
00005 
00006 
00007 
00008 #include "ActionSink_Proxy.hpp"
00009 
00010 namespace saftlib {
00011 
00026     class SoftwareActionSink_Proxy : public  ActionSink_Proxy   {
00027         static std::vector<std::string> gen_interface_names();
00028     public:
00029         SoftwareActionSink_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00030         static std::shared_ptr<SoftwareActionSink_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00031         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00046         // @saftbus-export
00047         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset);
00048     private:
00049         int interface_no;
00050 
00051     };
00052 
00053 }
00054 
00055 #endif
```

8.148 SoftwareActionSink_Service.hpp

```
00001 #ifndef SoftwareActionSink_SERVICE_HPP_
00002 #define SoftwareActionSink_SERVICE_HPP_
00003 
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006 
00007 #include <functional>
00008 
00009 namespace saftlib {
00010 
00011     class SoftwareActionSink;
00012     class SoftwareActionSink_Service : public saftbus::Service {
00013     public:
00014         SoftwareActionSink* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef SoftwareActionSink DriverType;
00017         SoftwareActionSink_Service(SoftwareActionSink* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         SoftwareActionSink_Service();
00019         ~SoftwareActionSink_Service();
```

```
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void OverflowCount_dispatch_function(uint64_t arg_1);
00023         sigc::connection OverflowCount_connection;
00024         void ActionCount_dispatch_function(uint64_t arg_1);
00025         sigc::connection ActionCount_connection;
00026         void LateCount_dispatch_function(uint64_t arg_1);
00027         sigc::connection LateCount_connection;
00028         void SigLate_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00029         sigc::connection SigLate_connection;
00030         void EarlyCount_dispatch_function(uint64_t arg_1);
00031         sigc::connection EarlyCount_connection;
00032         void SigEarly_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00033         sigc::connection SigEarly_connection;
00034         void ConflictCount_dispatch_function(uint64_t arg_1);
00035         sigc::connection ConflictCount_connection;
00036         void SigConflict_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00037         sigc::connection SigConflict_connection;
00038         void DelayedCount_dispatch_function(uint64_t arg_1);
00039         sigc::connection DelayedCount_connection;
00040         void SigDelayed_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00041         sigc::connection SigDelayed_connection;
00042         void Destroyed_dispatch_function();
00043         sigc::connection Destroyed_connection;
00044     };
00045
00046 }
00047
00048 #endif
```

8.149 SoftwareCondition.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021
00022 #ifndef SOFTWARE_CONDITION_H
00023 #define SOFTWARE_CONDITION_H
00024
00025 #include "Owned.hpp"
00026 #include "Condition.hpp"
00027
00028
00029 // @saftbus-include
00030 #include <Time.hpp>
00031 // @saftbus-include
00032 #include <sigc++/sigc++.h>
00033
00034 #include <saftbus/service.hpp>
00035
00036
00037 #include <functional>
00038
00039 namespace saftlib {
00040
00041
00042 class SoftwareCondition_Service;
00049 class SoftwareCondition : public Condition
00050 {
00051 public:
00052     SoftwareCondition(ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask,
      int64_t offset, saftbus::Container *container);
```

```
00053
00072     // @saftbus-export
00073     sigc::signal<void, uint64_t, uint64_t, saftlib::Time, saftlib::Time, uint16_t > SigAction;
00074
00075     // // @saftbus-export
00076     // std::function< void(uint64_t event, uint64_t param, saftlib::Time deadline, saftlib::Time
      executed, uint16_t flags) > Action;
00077
00078     typedef SoftwareCondition_Service ServiceType;
00079 };
00080
00081 }
00082
00083 #endif
```

8.150 SoftwareCondition_Proxy.hpp

```
00001 #ifndef SoftwareCondition_PROXY_HPP_
00002 #define SoftwareCondition_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006 #include <sigc++/sigc++.h>
00007
00008 #include <Time.hpp>
00009 #include <sigc++/sigc++.h>
00010
00011 #include "Condition_Proxy.hpp"
00012
00013 namespace saftlib {
00014
00021     class SoftwareCondition_Proxy : public  Condition_Proxy   {
00022         static std::vector<std::string> gen_interface_names();
00023     public:
00024         SoftwareCondition_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00025         static std::shared_ptr<SoftwareCondition_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00026         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00045     // @saftbus-export
00046         sigc::signal<void, uint64_t, uint64_t, saftlib::Time, saftlib::Time, uint16_t> SigAction;
00047     private:
00048         int interface_no;
00049
00050     };
00051
00052 }
00053
00054 #endif
```

8.151 SoftwareCondition_Service.hpp

```
00001 #ifndef SoftwareCondition_SERVICE_HPP_
00002 #define SoftwareCondition_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class SoftwareCondition;
00012     class SoftwareCondition_Service : public saftbus::Service {
00013     public:
00014         SoftwareCondition* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef SoftwareCondition DriverType;
00017         SoftwareCondition_Service(SoftwareCondition* instance, std::function<void()>
      destruction_callback = std::function<void()>(), bool destoy_if_owner_quits = true );
00018         SoftwareCondition_Service();
00019         ~SoftwareCondition_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void SigAction_dispatch_function(uint64_t arg_1,uint64_t arg_2,saftlib::Time
      arg_3,saftlib::Time arg_4,uint16_t arg_5);
00023         sigc::connection SigAction_connection;
```

```
00024         void Destroyed_dispatch_function();
00025         sigc::connection Destroyed_connection;
00026     };
00027
00028 }
00029
00030 #endif
```

8.152 TempSensor.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  ********************************************************************************
00004  *  This library is free software; you can redistribute it and/or
00005  *  modify it under the terms of the GNU Lesser General Public
00006  *  License as published by the Free Software Foundation; either
00007  *  version 3 of the License, or (at your option) any later version.
00008  *
00009  *  This library is distributed in the hope that it will be useful,
00010  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00012  *  Lesser General Public License for more details.
00013  *
00014  *  You should have received a copy of the GNU Lesser General Public
00015  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00016  ********************************************************************************
00017  */
00018
00019 #ifndef saftlib_TEMP_SENSOR_HPP_
00020 #define saftlib_TEMP_SENSOR_HPP_
00021
00022 #ifndef ETHERBONE_THROWS
00023 #define ETHERBONE_THROWS 1
00024 #define __STDC_FORMAT_MACROS
00025 #define __STDC_CONSTANT_MACROS
00026 #endif
00027 #include <etherbone.h>
00028
00029 #include "SdbDevice.hpp"
00030
00031 #include <map>
00032 #include <string>
00033
00034 namespace saftlib {
00035
00036 class TempSensor : public SdbDevice {
00037     mutable int32_t temperature;
00038 public:
00039     TempSensor(etherbone::Device &device);
00040
00046     // @saftbus-export
00047     bool getTemperatureSensorAvail() const;
00048
00055     // @saftbus-export
00056     int32_t CurrentTemperature();
00057
00058
00059 };
00060
00061
00062 }
00063
00064
00065 #endif
```

8.153 TempSensor_Proxy.hpp

```
00001 #ifndef TempSensor_PROXY_HPP_
00002 #define TempSensor_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008
00009 namespace saftlib {
00010
00011     class TempSensor_Proxy : public virtual saftbus::Proxy
00012     {
```

```
00013         static std::vector<std::string> gen_interface_names();
00014     public:
00015         TempSensor_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00016         static std::shared_ptr<TempSensor_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00017         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00023     // @saftbus-export
00024         bool getTemperatureSensorAvail();
00031     // @saftbus-export
00032         int32_t CurrentTemperature();
00033     private:
00034         int interface_no;
00035
00036     };
00037
00038 }
00039
00040 #endif
```

8.154 TempSensor_Service.hpp

```
00001 #ifndef TempSensor_SERVICE_HPP_
00002 #define TempSensor_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class TempSensor;
00012     class TempSensor_Service : public saftbus::Service {
00013     public:
00014         TempSensor* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef TempSensor DriverType;
00017         TempSensor_Service(TempSensor* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         TempSensor_Service();
00019         ~TempSensor_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022     };
00023
00024 }
00025
00026 #endif
```

8.155 Time.hpp

```
00001 /*  Copyright (C) 2020 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  ********************************************************************************
00004  *  This library is free software; you can redistribute it and/or
00005  *  modify it under the terms of the GNU Lesser General Public
00006  *  License as published by the Free Software Foundation; either
00007  *  version 3 of the License, or (at your option) any later version.
00008  *
00009  *  This library is distributed in the hope that it will be useful,
00010  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00012  *  Lesser General Public License for more details.
00013  *
00014  *  You should have received a copy of the GNU Lesser General Public
00015  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00016  ********************************************************************************
00017  */
00018
00020 // Conversion functions between UTC and TAI values
00021 //   (TAI epoch is 01/01/1970 within this library)
00022 // All time values are in nanoseconds
00023 // All functions handle negative leap seconds
00024 //   (in case they will be introduced in the future)
00026
00027 #ifndef SAFTLIB_TIME_H_
```

```
00028 #define SAFTLIB_TIME_H_
00029
00030 #include <stdint.h>
00031
00032
00033 namespace saftlib
00034 {
00035
00036     //extern const int64_t leap_second_list[][2];
00037     int64_t leap_second_epoch(int n);
00038     int64_t leap_second_offset(int n);
00039
00040     void init(const char* leap_second_list_filename = nullptr);
00041
00043     // convert TAI value to UTC value
00045     // parameters: TAI         nanosecond TAI value (nanoseconds since 01/01/1970)
00046     // returns:    UTC value that corresponds to the given TAI value. Result is
00047     //             ambiguous when it happens to be in a leap second interval. In
00048     //             that case, use the function TAI_is_UTCleap() to disambiguate.
00049     int64_t UTC_offset_TAI(uint64_t TAI);
00050     uint64_t TAI_to_UTC(uint64_t TAI);
00051
00053     // check if this TAI value falls into a UTC leap second interval
00055     // parameters: TAI         nanosecond TAI value (nanoseconds since 01/01/1970)
00056     // returns:    1 if this is a positive leap second interval (the UTC second
00057     //               part was repeated)
00058     //             0 if this was outside a leap second interval
00059     //            -1 if the previous second was skipped (only happens if we get
00060     //               negative leap seconds, which didn't happen yet (year 2019))
00061     int TAI_is_UTCleap(uint64_t TAI);
00062
00063
00065     // get UTC offset (in seconds) for given UTC value
00066     //    UTC_offset = TAI - UTC
00068     // parameters: UTC         nanosecond UTC value
00069     //             leap        1 if UTC1 is inside a leap second interval,
00070     //                         0 otherwise
00071     //                         (This parameter is only ever looked at, when the
00072     //                         UTC value is actually in a leap second interval.
00073     //                         For non leap second intervals, this parameter is
00074     //                         ignored.)
00075     //             offset      a pointer to where the resulting offset is copied
00076     //                         after successful computation (return value 1)
00077     //                         the offset is given in nanoseconds
00078     // returns:    1           if the conversion could be done,
00079     //             0           if the UTC value value was invalid
00080     //                         (nonexistent UTC values will only appear if we get
00081     //                         negative leap seconds, which didn't happen yet
00082     //                         (year 2019))
00083     int UTC_offset_UTC(uint64_t UTC, int leap, int64_t *offset);
00084
00086     // convert UTC value to TAI value
00088     // parameters: UTC         nanosecond UTC value
00089     //             leap        1 if UTC1 is inside a leap second interval,
00090     //                         0 otherwise
00091     //                         (This parameter is only ever looked at, when the
00092     //                         UTC value is actually in a leap second interval.
00093     //                         For non leap second intervals, this parameter is
00094     //                         ignored.)
00095     //             TAI         a pointer to where the result of the conversion is
00096     //                         copied after successful computation
00097     // returns:    1           if the conversion could be done,
00098     //             0           if the UTC value value was invalid
00099     //                         (nonexistent UTC values will only appear if we get
00100     //                         negative leap seconds, which didn't happen yet
00101     //                         (year 2019))
00102     int UTC_to_TAI(uint64_t UTC, int leap, uint64_t *TAI);
00103
00105     // calculate UTC time difference
00107     // parameters: UTC1        nanosecond UTC value
00108     //             leap1       1 if UTC1 is inside a leap second interval,
00109     //                         0 otherwise
00110     //                         (This parameter is only ever looked at, when the
00111     //                         UTC value is actually in a leap second interval.
00112     //                         For non leap second intervals, this parameter is
00113     //                         ignored.)
00114     //             UTC2        see UTC1
00115     //             leap2       see leap1
00116     //             difference  a pointer to where the number of nanoseconds
00117     //                         UTC1-UTC2 is stored (in case of return value 1)
00118     // returns:    1           if the difference could be calculated,
00119     //             0           if one of the UTC values was invalid
00120     //                         (nonexistent UTC values will only appear if we get
00121     //                         negative leap seconds, which didn't happen yet
00122     //                         (year 2019))
00123     int UTC_difference(uint64_t UTC1, int leap1,
00124                        uint64_t UTC2, int leap2,
```

```
00125                       int64_t *difference);
00126
00127
00128
00129
00130
00131
00132
00133
00134     // some test functions for the library function implementations
00135     void test_UTC_offset();
00136     void test_UTC_difference();
00137     void test_conversion_forth_and_back();
00138     void test_special_cases();
00139
00140
00141     const int64_t sec = INT64_C(1000000000);
00142     const int64_t msec = INT64_C(1000000);
00143     const int64_t usec = INT64_C(1000);
00144     const int64_t nsec = INT64_C(1);
00145
00146     class Time
00147     {
00148     public:
00149         // implicit conversion to uint64_t (do we really want this?)
00150         // operator uint64_t() {
00151         //  return TAI;
00152         // }
00153
00154     public:
00155         Time() : TAI(0) {
00156         }
00157         Time(const Time& t) : TAI(t.TAI) {
00158         }
00159         uint64_t getUTC() const {
00160             return TAI_to_UTC(TAI);
00161         }
00162         uint64_t getTAI() const {
00163             return TAI;
00164         }
00165         Time& operator=(const Time& t) {
00166             TAI = t.TAI;
00167             return *this;
00168         }
00169         Time& operator+=(const int64_t& duration) {
00170             TAI += duration;
00171             return *this;
00172         }
00173         Time& operator-=(const int64_t& duration) {
00174             TAI -= duration;
00175             return *this;
00176         }
00177         int64_t operator-(const Time& rhs) const {
00178             return TAI - rhs.TAI;
00179         }
00180         int64_t getUTCOffset() const {
00181             return UTC_offset_TAI(TAI);
00182         }
00183         bool operator>(const Time& rhs) const {
00184             return TAI > rhs.TAI;
00185         }
00186         bool operator<(const Time& rhs) const {
00187             return TAI < rhs.TAI;
00188         }
00189         bool operator>=(const Time& rhs) const {
00190             return TAI >= rhs.TAI;
00191         }
00192         bool operator<=(const Time& rhs) const {
00193             return TAI <= rhs.TAI;
00194         }
00195         bool operator==(const Time& rhs) const {
00196             return TAI == rhs.TAI;
00197         }
00198         bool operator!=(const Time& rhs) const {
00199             return TAI != rhs.TAI;
00200         }
00201         int isLeapUTC() const {
00202             return TAI_is_UTCleap(TAI);
00203         }
00204
00205         friend Time makeTimeUTC(uint64_t UTC, bool isLeap);
00206         friend Time makeTimeTAI(uint64_t TAI);
00207     private:
00208         explicit Time(const uint64_t& value) : TAI(value) {
00209         }
00210
00211         uint64_t TAI;
```

```
00212     };
00213
00214     Time operator+(const Time& lhs, const int64_t& rhs);
00215     Time operator-(const Time& lhs, const int64_t& rhs);
00216     Time operator+(const int64_t& lhs, const Time& rhs);
00217     Time operator-(const int64_t& lhs, const Time& rhs);
00218     Time makeTimeUTC(uint64_t UTC, bool isLeap = false);
00219     Time makeTimeTAI(uint64_t TAI);
00220
00221
00222 }
00223
00224
00225 #endif
```

8.156 TimingReceiver.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_TIMING_RECEIVER_HPP_
00023 #define saftlib_TIMING_RECEIVER_HPP_
00024
00025 #include <deque>
00026 #include <memory>
00027 #include <string>
00028
00029 #include <saftbus/loop.hpp>
00030 #include <saftbus/service.hpp>
00031
00032 // direct base classes of TimingReceiver
00033 #include "OpenDevice.hpp"
00034 #include "Watchdog.hpp"
00035 #include "WhiteRabbit.hpp"
00036 #include "ECA.hpp"
00037 #include "ECA_TLU.hpp"
00038 #include "ECA_Event.hpp"
00039 #include "BuildIdRom.hpp"
00040 #include "TempSensor.hpp"
00041 #include "Reset.hpp"
00042 #include "Mailbox.hpp"
00043 #include "LM32Cluster.hpp"
00044
00045
00046 // classes used by TimingReceiver
00047 #include "IoControl.hpp"
00048
00049 #include "TimingReceiverAddon.hpp"
00050
00051 // @saftbus-include
00052 #include <Time.hpp>
00053
00054 namespace saftlib {
00055
00056 class SAFTd;
00057
00100 class TimingReceiver : public OpenDevice
00101                      , public Watchdog
00102                      , public WhiteRabbit
00103                      , public ECA
00104                      , public ECA_TLU
00105                      , public ECA_Event
00106                      , public BuildIdRom
00107                      , public TempSensor
00108                      , public Reset
```

```
00109                     , public Mailbox
00110                     , public LM32Cluster {
00111 public:
00112     TimingReceiver(SAFTd &saftd, const std::string &name, const std::string &etherbone_path,
00113                    int polling_interval_ms = 1, saftbus::Container *container = nullptr);
00114     ~TimingReceiver();
00115
00116     const std::string &getObjectPath() const;
00117
00120     // @saftbus-export
00121     void Remove();
00122
00126     // @saftbus-export
00127     std::string getName() const;
00128
00138     // @saftbus-export
00139     saftlib::Time CurrentTime() const;
00140
00150     // @saftbus-export
00151     void InjectEvent(uint64_t event, uint64_t param, saftlib::Time time) const;
00152
00153
00160     // @saftbus-export
00161     std::map< std::string, std::map< std::string, std::string > > getInterfaces() const;
00162
00167     // void addInterfaces(const std::string &interface_name, const std::map< std::string, std::string
      > & objects);
00168
00169     void installAddon(const std::string &interface_name, std::unique_ptr<TimingReceiverAddon> addon);
00170
00171     void removeAddon(const std::string &interface_name);
00172
00173 private:
00174     saftbus::Container *container;
00175
00176     IoControl io_control;
00177
00178     bool poll();
00179     saftbus::SourceHandle poll_timeout_source;
00180
00181
00182     eb_address_t ats;
00183
00184     std::string object_path;
00185     std::string name;
00186
00187
00188
00189     bool activate_msi_polling;
00190     unsigned polling_interval_ms;
00191
00192
00193     // TimingReceiver doesn't own TimingreceiverAddons
00194     std::map<std::string, std::unique_ptr<TimingReceiverAddon> > addons;
00195
00196 };
00197
00198 } // namespace
00199
00200 #endif
```

8.157 TimingReceiver_Proxy.hpp

```
00001 #ifndef TimingReceiver_PROXY_HPP_
00002 #define TimingReceiver_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007 #include <Time.hpp>
00008
00009 #include "BuildIdRom_Proxy.hpp"
00010 #include "ECA_Proxy.hpp"
00011 #include "ECA_Event_Proxy.hpp"
00012 #include "ECA_TLU_Proxy.hpp"
00013 #include "LM32Cluster_Proxy.hpp"
00014 #include "OpenDevice_Proxy.hpp"
00015 #include "Reset_Proxy.hpp"
00016 #include "TempSensor_Proxy.hpp"
00017 #include "WhiteRabbit_Proxy.hpp"
00018
00019 namespace saftlib {
00020
```

```
00063     class TimingReceiver_Proxy : public  BuildIdRom_Proxy , public  ECA_Proxy , public
      ECA_Event_Proxy , public  ECA_TLU_Proxy , public  LM32Cluster_Proxy , public  OpenDevice_Proxy ,
      public  Reset_Proxy , public  TempSensor_Proxy , public  WhiteRabbit_Proxy   {
00064         static std::vector<std::string> gen_interface_names();
00065     public:
00066         TimingReceiver_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00067         static std::shared_ptr<TimingReceiver_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00068         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00071     // @saftbus-export
00072         void Remove();
00076     // @saftbus-export
00077         std::string getName();
00087     // @saftbus-export
00088         saftlib::Time CurrentTime();
00098     // @saftbus-export
00099         void InjectEvent(uint64_t event, uint64_t param, saftlib::Time time);
00106     // @saftbus-export
00107         std::map< std::string, std::map< std::string, std::string > > getInterfaces();
00108     private:
00109         int interface_no;
00110
00111     };
00112
00113 }
00114
00115 #endif
```

8.158 TimingReceiver_Service.hpp

```
00001 #ifndef TimingReceiver_SERVICE_HPP_
00002 #define TimingReceiver_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class TimingReceiver;
00012     class TimingReceiver_Service : public saftbus::Service {
00013     public:
00014         TimingReceiver* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef TimingReceiver DriverType;
00017         TimingReceiver_Service(TimingReceiver* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         TimingReceiver_Service();
00019         ~TimingReceiver_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Locked_dispatch_function(bool arg_1);
00023         sigc::connection Locked_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.159 TimingReceiverAddon.hpp

```
00001 #ifndef SAFTLIB_TIMING_RECEIVER_ADDON_HPP_
00002 #define SAFTLIB_TIMING_RECEIVER_ADDON_HPP_
00003
00004 #include <string>
00005 #include <map>
00006
00007 namespace saftlib {
00008
00009     class TimingReceiverAddon
00010     {
00011     public:
00012         TimingReceiverAddon();
00013         virtual ~TimingReceiverAddon();
00014
```

```
00015         virtual std::map< std::string, std::map< std::string, std::string> > getObjects() = 0;
00016     };
00017
00018
00019
00020 }
00021
00022 #endif
```

8.160 Watchdog.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef saftlib_WATCHDOG_HPP_
00023 #define saftlib_WATCHDOG_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "SdbDevice.hpp"
00033
00034 namespace saftlib {
00035
00036 class Watchdog : public SdbDevice {
00037     eb_data_t watchdog_value;
00038 public:
00039     Watchdog(etherbone::Device &device);
00040     bool aquire();
00041     void update();
00042 };
00043
00044 }
00045
00046 #endif
```

8.161 WbmActionSink.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
```

```
00020  */
00021
00022 #ifndef SAFTLIB_WBM_ACTION_SINK_HPP_
00023 #define SAFTLIB_WBM_ACTION_SINK_HPP_
00024
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
00030 #include <etherbone.h>
00031
00032 #include "ActionSink.hpp"
00033
00034 namespace saftlib {
00035
00045 class WbmActionSink : public ActionSink {
00046     public:
00047         WbmActionSink(etherbone::Device &device
00048                       , ECA &eca
00049                       , const std::string &object_path
00050                       , const std::string &name
00051                       , unsigned channel, eb_address_t acwbm_address
00052                       , saftbus::Container *container = nullptr);
00053
00069         // @saftbus-export
00070         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t
      tag);
00071
00072
00073     // @saftbus-export
00074     void ExecuteMacro(uint32_t idx);
00075     // @saftbus-export
00076     void RecordMacro(uint32_t idx, const std::vector< std::vector< uint32_t > >& commands);
00077     // @saftbus-export
00078     void ClearMacro(uint32_t idx);
00079     // @saftbus-export
00080     void ClearAllMacros();
00081     // Property getters
00082     // @saftbus-export
00083     unsigned char getStatus() const;
00084     // @saftbus-export
00085     uint32_t getMaxMacros() const;
00086     // @saftbus-export
00087     uint32_t getMaxSpace() const;
00088     // @saftbus-export
00089     bool getEnable() const;
00090     // @saftbus-export
00091     uint32_t getLastExecutedIdx() const;
00092     // @saftbus-export
00093     uint32_t getLastRecordedIdx() const;
00094     // Property setters
00095     // @saftbus-export
00096     void setEnable(bool val);
00097
00098     protected:
00099         etherbone::Device &device;
00100         eb_address_t acwbm;
00101 };
00102
00103 }
00104
00105 #endif
```

8.162 WbmActionSink_Proxy.hpp

```
00001 #ifndef WbmActionSink_PROXY_HPP_
00002 #define WbmActionSink_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "ActionSink_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00021     class WbmActionSink_Proxy : public  ActionSink_Proxy     {
00022         static std::vector<std::string> gen_interface_names();
00023     public:
00024         WbmActionSink_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
```

```
00025         static std::shared_ptr<WbmActionSink_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00026         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00042         // @saftbus-export
00043         std::string NewCondition(bool active, uint64_t id, uint64_t mask, int64_t offset, uint32_t
      tag);
00044     // @saftbus-export
00045         void ExecuteMacro(uint32_t idx);
00046     // @saftbus-export
00047         void RecordMacro(uint32_t idx, const std::vector< std::vector< uint32_t > >& commands);
00048     // @saftbus-export
00049         void ClearMacro(uint32_t idx);
00050     // @saftbus-export
00051         void ClearAllMacros();
00052     // Property getters
00053     // @saftbus-export
00054         unsigned char getStatus();
00055     // @saftbus-export
00056         uint32_t getMaxMacros();
00057     // @saftbus-export
00058         uint32_t getMaxSpace();
00059     // @saftbus-export
00060         bool getEnable();
00061     // @saftbus-export
00062         uint32_t getLastExecutedIdx();
00063     // @saftbus-export
00064         uint32_t getLastRecordedIdx();
00065     // Property setters
00066     // @saftbus-export
00067         void setEnable(bool val);
00068     private:
00069         int interface_no;
00070
00071     };
00072
00073 }
00074
00075 #endif
```

8.163 WbmActionSink_Service.hpp

```
00001 #ifndef WbmActionSink_SERVICE_HPP_
00002 #define WbmActionSink_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class WbmActionSink;
00012     class WbmActionSink_Service : public saftbus::Service {
00013     public:
00014         WbmActionSink* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef WbmActionSink DriverType;
00017         WbmActionSink_Service(WbmActionSink* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         WbmActionSink_Service();
00019         ~WbmActionSink_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void OverflowCount_dispatch_function(uint64_t arg_1);
00023         sigc::connection OverflowCount_connection;
00024         void ActionCount_dispatch_function(uint64_t arg_1);
00025         sigc::connection ActionCount_connection;
00026         void LateCount_dispatch_function(uint64_t arg_1);
00027         sigc::connection LateCount_connection;
00028         void SigLate_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00029         sigc::connection SigLate_connection;
00030         void EarlyCount_dispatch_function(uint64_t arg_1);
00031         sigc::connection EarlyCount_connection;
00032         void SigEarly_dispatch_function(uint32_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00033         sigc::connection SigEarly_connection;
00034         void ConflictCount_dispatch_function(uint64_t arg_1);
00035         sigc::connection ConflictCount_connection;
00036         void SigConflict_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00037         sigc::connection SigConflict_connection;
```

```
00038         void DelayedCount_dispatch_function(uint64_t arg_1);
00039         sigc::connection DelayedCount_connection;
00040         void SigDelayed_dispatch_function(uint64_t arg_1,uint64_t arg_2,uint64_t arg_3,saftlib::Time
      arg_4,saftlib::Time arg_5);
00041         sigc::connection SigDelayed_connection;
00042         void Destroyed_dispatch_function();
00043         sigc::connection Destroyed_connection;
00044     };
00045
00046 }
00047
00048 #endif
```

8.164 WbmCondition.hpp

```
00001 /*  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ********************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ********************************************************************************
00020  */
00021
00022 #ifndef WBM_CONDITION_HPP_
00023 #define WBM_CONDITION_HPP_
00024
00025 #include <saftbus/service.hpp>
00026
00027 #include "Owned.hpp"
00028 #include "Condition.hpp"
00029 #include "ActionSink.hpp"
00030
00031 namespace saftlib {
00032
00033
00034 class WbmCondition_Service;
00035
00036
00043 class WbmCondition : public Condition {
00044 public:
00045     WbmCondition(ActionSink *sink, unsigned number, bool active, uint64_t id, uint64_t mask, int64_t
      offset, uint32_t tag, saftbus::Container *container = nullptr);
00046
00047     typedef WbmCondition_Service ServiceType;
00048 };
00049
00050 }
00051
00052 #endif
```

8.165 WbmCondition_Proxy.hpp

```
00001 #ifndef WbmCondition_PROXY_HPP_
00002 #define WbmCondition_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006
00007
00008 #include "Condition_Proxy.hpp"
00009
00010 namespace saftlib {
00011
00018     class WbmCondition_Proxy : public  Condition_Proxy     {
00019         static std::vector<std::string> gen_interface_names();
00020     public:
```

```
00021         WbmCondition_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00022         static std::shared_ptr<WbmCondition_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00023         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00024     private:
00025         int interface_no;
00026 
00027     };
00028 
00029 }
00030 
00031 #endif
```

8.166 WbmCondition_Service.hpp

```
00001 #ifndef WbmCondition_SERVICE_HPP_
00002 #define WbmCondition_SERVICE_HPP_
00003 
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006 
00007 #include <functional>
00008 
00009 namespace saftlib {
00010 
00011     class WbmCondition;
00012     class WbmCondition_Service : public saftbus::Service {
00013     public:
00014         WbmCondition* d;
00015         static std::vector<std::string> gen_interface_names();
00016         typedef WbmCondition DriverType;
00017         WbmCondition_Service(WbmCondition* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         WbmCondition_Service();
00019         ~WbmCondition_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021 
00022         void Destroyed_dispatch_function();
00023         sigc::connection Destroyed_connection;
00024     };
00025 
00026 }
00027 
00028 #endif
```

8.167 WhiteRabbit.hpp

```
00001 /* *  Copyright (C) 2011-2016, 2021-2022 GSI Helmholtz Centre for Heavy Ion Research GmbH
00002  *
00003  *  @author Wesley W. Terpstra <w.terpstra@gsi.de>
00004  *          Michael Reese <m.reese@gsi.de>
00005  *
00006  ******************************************************************************
00007  *  This library is free software; you can redistribute it and/or
00008  *  modify it under the terms of the GNU Lesser General Public
00009  *  License as published by the Free Software Foundation; either
00010  *  version 3 of the License, or (at your option) any later version.
00011  *
00012  *  This library is distributed in the hope that it will be useful,
00013  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00015  *  Lesser General Public License for more details.
00016  *
00017  *  You should have received a copy of the GNU Lesser General Public
00018  *  License along with this library. If not, see <http://www.gnu.org/licenses/>.
00019  ******************************************************************************
00020  */
00021 
00022 #ifndef saftlib_WHITE_RABBIT_HPP_
00023 #define saftlib_WHITE_RABBIT_HPP_
00024 
00025 #ifndef ETHERBONE_THROWS
00026 #define ETHERBONE_THROWS 1
00027 #define __STDC_FORMAT_MACROS
00028 #define __STDC_CONSTANT_MACROS
00029 #endif
```

```
00030 #include <etherbone.h>
00031
00032 // @saftbus-export
00033 #include <sigc++/sigc++.h>
00034
00035 #include "SdbDevice.hpp"
00036
00037 namespace saftlib {
00038
00039 class WhiteRabbit : public SdbDevice {
00040 protected:
00041     mutable bool locked;
00042 public:
00043     WhiteRabbit(etherbone::Device &device);
00044
00051     // @saftbus-export
00052     bool getLocked() const;
00053
00054     // @saftbus-export
00055     sigc::signal<void, bool> Locked;
00056 };
00057
00058 }
00059
00060 #endif
```

8.168 WhiteRabbit_Proxy.hpp

```
00001 #ifndef WhiteRabbit_PROXY_HPP_
00002 #define WhiteRabbit_PROXY_HPP_
00003
00004 #include <saftbus/client.hpp>
00005
00006 #include <sigc++/sigc++.h>
00007
00008 #include <sigc++/sigc++.h>
00009
00010
00011 namespace saftlib {
00012
00013     class WhiteRabbit_Proxy : public virtual saftbus::Proxy
00014     {
00015         static std::vector<std::string> gen_interface_names();
00016     public:
00017         WhiteRabbit_Proxy(const std::string &object_path, saftbus::SignalGroup &signal_group =
      saftbus::SignalGroup::get_global(), const std::vector<std::string> &interface_names =
      gen_interface_names());
00018         static std::shared_ptr<WhiteRabbit_Proxy> create(const std::string &object_path,
      saftbus::SignalGroup &signal_group = saftbus::SignalGroup::get_global());
00019         bool signal_dispatch(int interface_no, int signal_no, saftbus::Deserializer &signal_content);
00026     // @saftbus-export
00027         bool getLocked();
00028     // @saftbus-export
00029         sigc::signal<void, bool> Locked;
00030     private:
00031         int interface_no;
00032
00033     };
00034
00035 }
00036
00037 #endif
```

8.169 WhiteRabbit_Service.hpp

```
00001 #ifndef WhiteRabbit_SERVICE_HPP_
00002 #define WhiteRabbit_SERVICE_HPP_
00003
00004 #include <saftbus/service.hpp>
00005 #include <saftbus/saftbus.hpp>
00006
00007 #include <functional>
00008
00009 namespace saftlib {
00010
00011     class WhiteRabbit;
00012     class WhiteRabbit_Service : public saftbus::Service {
00013     public:
00014         WhiteRabbit* d;
```

```
00015         static std::vector<std::string> gen_interface_names();
00016         typedef WhiteRabbit DriverType;
00017         WhiteRabbit_Service(WhiteRabbit* instance, std::function<void()> destruction_callback =
      std::function<void()>(), bool destoy_if_owner_quits = true );
00018         WhiteRabbit_Service();
00019         ~WhiteRabbit_Service();
00020         void call(unsigned interface_no, unsigned function_no, int client_fd, saftbus::Deserializer
      &received, saftbus::Serializer &send);
00021
00022         void Locked_dispatch_function(bool arg_1);
00023         sigc::connection Locked_connection;
00024     };
00025
00026 }
00027
00028 #endif
```

8.170 wr_mil_gw_regs.h

```
00001 #ifndef WR_MIL_GW_H_
00002 #define WR_MIL_GW_H_
00003
00004 // Magic number to identify the LM32 (if more than one exists) that runs the WR-MIL gateway
00005 #define WR_MIL_GW_MAGIC_NUMBER       0x1234abcd
00006
00007 #define WR_MIL_GW_SHARED_OFFSET      0x500 // offset to the shared memory section
00008
00009 // Commands to be written into the WR_MIL_GW_REG_COMMAND register
00010 #define WR_MIL_GW_CMD_NONE           0x0   // empty command
00011 #define WR_MIL_GW_CMD_KILL           0x1   // command to stop the LM32 from running
00012 #define WR_MIL_GW_CMD_RESET          0x2   // command to stop the LM32 for 1sec and go into initial
      state
00013 #define WR_MIL_GW_CMD_CONFIG_SIS     0x3   // command to configure the gateway for SIS operation
00014 #define WR_MIL_GW_CMD_CONFIG_ESR     0x4   // command to configure the gateway for ESR operation
00015 #define WR_MIL_GW_CMD_TEST           0x5   // command that does nothing. it is useful to initiate to
      see if the firmware cleans the command register (see if the firmware runs at all)
00016 #define WR_MIL_GW_CMD_UPDATE_OLED    0x6   // redraw content on OLED
00017
00018
00019 // Configuration register mapping in shared memory region
00020 #define WR_MIL_GW_REG_MAGIC_NUMBER   0x00  // command to be executed
00021 #define WR_MIL_GW_REG_COMMAND        0x04  // command to be executed
00022 #define WR_MIL_GW_REG_UTC_TRIGGER    0x08  // the MIL event that triggers the generation of UTC events
00023 #define WR_MIL_GW_REG_UTC_DELAY      0x0C  // delay [us] between the 5 generated UTC MIL events
00024 #define WR_MIL_GW_REG_TRIG_UTC_DELAY 0x10  // delay [us] between the trigger event and the first UTC
      (and other) generated events
00025 #define WR_MIL_GW_REG_EVENT_SOURCE   0x14  // for internal use: register to hold the source
      configuration: 1 = SIS ; 2 = ESR ; 0 not configured
00026 #define WR_MIL_GW_REG_LATENCY        0x18  // MIL event is generated 100us+latency after the WR event.
      The value of latency can be negative
00027 #define WR_MIL_GW_REG_STATE          0x1C  // for internal use: state of the program: INITIAL,
      UNCONFIGURED, CONFIGURED
00028 #define WR_MIL_GW_REG_UTC_OFFSET_HI  0x20  // delay [ms] between the TAI and the MIL-UTC, high word
00029 #define WR_MIL_GW_REG_UTC_OFFSET_LO  0x24  // delay [ms] between the trigger the MIL-UTC, low  word
00030 #define WR_MIL_GW_REG_NUM_EVENTS_HI  0x28  // number of translated events from WR to MIL, high word
00031 #define WR_MIL_GW_REG_NUM_EVENTS_LO  0x2C  // number of translated events from WR to MIL, low word
00032 #define WR_MIL_GW_REG_LATE_EVENTS    0x30  // number of translated events that could not be delivered
      in time
00033 #define WR_MIL_GW_REG_LATE_HISTOGRAM 0x34  // dummy register to indicate position after the last valid
      register
00034 #define WR_MIL_GW_REG_MIL_HISTOGRAM  0x74  // dummy register to indicate position after the last valid
      register
00035 #define WR_MIL_GW_REG_MSI_SLOT       0x474 // MSI slot is stored here
00036 #define WR_MIL_GW_REG_SET_OP_READY   0x478 // Host writes 1 if OP-READY, 0 otherwise
00037 #define WR_MIL_GW_REG_REQUEST_FILL_EVT 0x47c // if this is written to 1, the gateway will send a fill
      event as soon as possible
00038
00039 // states of the software
00040 #define WR_MIL_GW_STATE_INIT         0
00041 #define WR_MIL_GW_STATE_UNCONFIGURED 1
00042 #define WR_MIL_GW_STATE_CONFIGURED   2
00043 #define WR_MIL_GW_STATE_PAUSED       3
00044
00045 // Constants for event source type configuration.
00046 // The WR-MIL gateway can run the SIS or the ESR
00047 // and has to be configured to one these.
00048 // At startup it is unconfigured.
00049 #define WR_MIL_GW_EVENT_SOURCE_UNKNOWN 0x0
00050 #define WR_MIL_GW_EVENT_SOURCE_SIS     0x1
00051 #define WR_MIL_GW_EVENT_SOURCE_ESR     0x2
00052
00053 // Interrupts
00054 #define WR_MIL_GW_MSI_LATE_EVENT       0x1
```

```
00055 #define WR_MIL_GW_MSI_STATE_CHANGED    0x2
00056
00057 #endif
```

Index