SoapySDR::Device Class Reference

`#include <Device.hpp>`

Public Member Functions

virtual	~Device (void) virtual destructor for inheritance More...
virtual std::string	getDriverKey (void) const
virtual std::string	getHardwareKey (void) const
virtual Kwargs	getHardwareInfo (void) const
virtual void	setFrontendMapping (const int direction, const std::string &mapping)
virtual std::string	getFrontendMapping (const int direction) const
virtual size_t	getNumChannels (const int direction) const
virtual Kwargs	getChannelInfo (const int direction, const size_t channel) const
virtual bool	getFullDuplex (const int direction, const size_t channel) const
virtual std::vector< std::string >	getStreamFormats (const int direction, const size_t channel) const
virtual std::string	getNativeStreamFormat (const int direction, const size_t channel, double &fullScale) const
virtual ArgInfoList	getStreamArgsInfo (const int direction, const size_t channel) const
virtual Stream *	setupStream (const int direction, const std::string &format, const std::vector< size_t > &channels=std::vector< size_t >(), const Kwargs &args=Kwargs())
virtual void	closeStream (Stream *stream)
virtual size_t	getStreamMTU (Stream *stream) const
virtual int	activateStream (Stream *stream, const int flags=0, const long long timeNs=0, const size_t numElems=0)
virtual int	deactivateStream (Stream *stream, const int flags=0, const long long timeNs=0)
virtual int	readStream (Stream *stream, void *const *buffs, const size_t numElems, int &flags, long long &timeNs, const long timeoutUs=100000)
virtual int	writeStream (Stream *stream, const void *const *buffs, const size_t numElems, int &flags, const long long timeNs=0, const long timeoutUs=100000)
virtual int	readStreamStatus (Stream *stream, size_t &chanMask, int &flags, long long &timeNs, const long timeoutUs=100000)
virtual size_t	getNumDirectAccessBuffers (Stream *stream)
virtual int	getDirectAccessBufferAddrs (Stream *stream, const size_t handle, void **buffs)
virtual int	acquireReadBuffer (Stream *stream, size_t &handle, const void **buffs, int &flags, long long &timeNs, const long timeoutUs=100000)
virtual void	releaseReadBuffer (Stream *stream, const size_t handle)

virtual int	acquireWriteBuffer (Stream *stream, size_t &handle, void **buffs, const long timeoutUs=100000)
virtual void	releaseWriteBuffer (Stream *stream, const size_t handle, const size_t numElems, int &flags, const long long timeNs=0)
virtual std::vector< std::string >	listAntennas (const int direction, const size_t channel) const
virtual void	setAntenna (const int direction, const size_t channel, const std::string &name)
virtual std::string	getAntenna (const int direction, const size_t channel) const
virtual bool	hasDCOffsetMode (const int direction, const size_t channel) const
virtual void	setDCOffsetMode (const int direction, const size_t channel, const bool automatic)
virtual bool	getDCOffsetMode (const int direction, const size_t channel) const
virtual bool	hasDCOffset (const int direction, const size_t channel) const
virtual void	setDCOffset (const int direction, const size_t channel, const std::complex< double > &offset)
virtual std::complex< double >	getDCOffset (const int direction, const size_t channel) const
virtual bool	hasIQBalance (const int direction, const size_t channel) const
virtual void	setIQBalance (const int direction, const size_t channel, const std::complex< double > &balance)
virtual std::complex< double >	getIQBalance (const int direction, const size_t channel) const
virtual bool	hasIQBalanceMode (const int direction, const size_t channel) const
virtual void	setIQBalanceMode (const int direction, const size_t channel, const bool automatic)
virtual bool	getIQBalanceMode (const int direction, const size_t channel) const
virtual bool	hasFrequencyCorrection (const int direction, const size_t channel) const
virtual void	setFrequencyCorrection (const int direction, const size_t channel, const double value)
virtual double	getFrequencyCorrection (const int direction, const size_t channel) const
virtual std::vector< std::string >	listGains (const int direction, const size_t channel) const
virtual bool	hasGainMode (const int direction, const size_t channel) const
virtual void	setGainMode (const int direction, const size_t channel, const bool automatic)
virtual bool	getGainMode (const int direction, const size_t channel) const
virtual void	setGain (const int direction, const size_t channel, const double value)
virtual void	setGain (const int direction, const size_t channel, const std::string &name, const double value)
virtual double	getGain (const int direction, const size_t channel) const
virtual double	getGain (const int direction, const size_t channel, const std::string &name) const
virtual Range	getGainRange (const int direction, const size_t channel) const
virtual Range	getGainRange (const int direction, const size_t channel, const std::string &name) const
virtual void	setFrequency (const int direction, const size_t channel, const double frequency, const Kwargs &args=Kwargs())

virtual void	setFrequency (const int direction, const size_t channel, const std::string &name, const double frequency, const Kwargs &args=Kwargs())
virtual double	getFrequency (const int direction, const size_t channel) const
virtual double	getFrequency (const int direction, const size_t channel, const std::string &name) const
virtual std::vector< std::string >	listFrequencies (const int direction, const size_t channel) const
virtual RangeList	getFrequencyRange (const int direction, const size_t channel) const
virtual RangeList	getFrequencyRange (const int direction, const size_t channel, const std::string &name) const
virtual ArgInfoList	getFrequencyArgsInfo (const int direction, const size_t channel) const
virtual void	setSampleRate (const int direction, const size_t channel, const double rate)
virtual double	getSampleRate (const int direction, const size_t channel) const
virtual std::vector< double >	listSampleRates (const int direction, const size_t channel) const
virtual RangeList	getSampleRateRange (const int direction, const size_t channel) const
virtual void	setBandwidth (const int direction, const size_t channel, const double bw)
virtual double	getBandwidth (const int direction, const size_t channel) const
virtual std::vector< double >	listBandwidths (const int direction, const size_t channel) const
virtual RangeList	getBandwidthRange (const int direction, const size_t channel) const
virtual void	setMasterClockRate (const double rate)
virtual double	getMasterClockRate (void) const
virtual RangeList	getMasterClockRates (void) const
virtual void	setReferenceClockRate (const double rate)
virtual double	getReferenceClockRate (void) const
virtual RangeList	getReferenceClockRates (void) const
virtual std::vector< std::string >	listClockSources (void) const
virtual void	setClockSource (const std::string &source)
virtual std::string	getClockSource (void) const
virtual std::vector< std::string >	listTimeSources (void) const
virtual void	setTimeSource (const std::string &source)
virtual std::string	getTimeSource (void) const
virtual bool	hasHardwareTime (const std::string &what="") const
virtual long long	getHardwareTime (const std::string &what="") const
virtual void	setHardwareTime (const long long timeNs, const std::string &what="")
virtual void	setCommandTime (const long long timeNs, const std::string &what="")
virtual std::vector< std::string >	listSensors (void) const
virtual ArgInfo	getSensorInfo (const std::string &key) const
virtual std::string	readSensor (const std::string &key) const
template<typename Type >	
Type	readSensor (const std::string &key) const

virtual std::vector< std::string >	listSensors (const int direction, const size_t channel) const
virtual ArgInfo	getSensorInfo (const int direction, const size_t channel, const std::string &key) const
virtual std::string	readSensor (const int direction, const size_t channel, const std::string &key) const
template<typename Type >	
Type	readSensor (const int direction, const size_t channel, const std::string &key) const
virtual std::vector< std::string >	listRegisterInterfaces (void) const
virtual void	writeRegister (const std::string &name, const unsigned addr, const unsigned value)
virtual unsigned	readRegister (const std::string &name, const unsigned addr) const
virtual void	writeRegister (const unsigned addr, const unsigned value)
virtual unsigned	readRegister (const unsigned addr) const
virtual void	writeRegisters (const std::string &name, const unsigned addr, const std::vector< unsigned > &value)
virtual std::vector< unsigned >	readRegisters (const std::string &name, const unsigned addr, const size_t length) const
virtual ArgInfoList	getSettingInfo (void) const
virtual void	writeSetting (const std::string &key, const std::string &value)
template<typename Type >	
void	writeSetting (const std::string &key, const Type &value)
virtual std::string	readSetting (const std::string &key) const
template<typename Type >	
Type	readSetting (const std::string &key)
virtual ArgInfoList	getSettingInfo (const int direction, const size_t channel) const
virtual void	writeSetting (const int direction, const size_t channel, const std::string &key, const std::string &value)
template<typename Type >	
void	writeSetting (const int direction, const size_t channel, const std::string &key, const Type &value)
virtual std::string	readSetting (const int direction, const size_t channel, const std::string &key) const
template<typename Type >	
Type	readSetting (const int direction, const size_t channel, const std::string &key)
virtual std::vector< std::string >	listGPIOBanks (void) const
virtual void	writeGPIO (const std::string &bank, const unsigned value)
virtual void	writeGPIO (const std::string &bank, const unsigned value, const unsigned mask)
virtual unsigned	readGPIO (const std::string &bank) const
virtual void	writeGPIODir (const std::string &bank, const unsigned dir)
virtual void	writeGPIODir (const std::string &bank, const unsigned dir, const unsigned mask)
virtual unsigned	readGPIODir (const std::string &bank) const
virtual void	writeI2C (const int addr, const std::string &data)

virtual std::string	readI2C (const int addr, const size_t numBytes)
virtual unsigned	transactSPI (const int addr, const unsigned data, const size_t numBits)
virtual std::vector< std::string >	listUARTs (void) const
virtual void	writeUART (const std::string &which, const std::string &data)
virtual std::string	readUART (const std::string &which, const long timeoutUs=100000) const
virtual void *	getNativeDeviceHandle (void) const

Static Public Member Functions

static KwargsList	enumerate (const Kwargs &args=Kwargs())
static KwargsList	enumerate (const std::string &args)
static Device *	make (const Kwargs &args=Kwargs())
static Device *	make (const std::string &args)
static void	unmake (Device *device)
static std::vector< Device * >	make (const KwargsList &argsList)
static std::vector< Device * >	make (const std::vector< std::string > &argsList)
static void	unmake (const std::vector< Device * > &devices)

Detailed Description

Abstraction for an SDR tranceiver device - configuration and streaming.

Constructor & Destructor Documentation

◆ ~Device()

virtual SoapySDR::Device::~Device (void) virtual

virtual destructor for inheritance

Member Function Documentation

◆ acquireReadBuffer()

```
virtual int SoapySDR::Device::acquireReadBuffer ( Stream *       stream,
                                                  size_t &       handle,
                                                  const void **  buffs,
                                                  int &          flags,
                                                  long long &    timeNs,
                                                  const long     timeoutUs = 100000
                                                )                                    virtual
```

Acquire direct buffers from a receive stream. This call is part of the direct buffer access API.

The buffs array will be filled with a stream pointer for each channel. Each pointer can be read up to the number of return value elements.

The handle will be set by the implementation so that the caller may later release access to the buffers with releaseReadBuffer(). Handle represents an index into the internal scatter/gather table such that handle is between 0 and num direct buffers - 1.

Parameters

stream	the opaque pointer to a stream handle
handle	an index value used in the release() call
buffs	an array of void* buffers num chans in size
flags	optional flag indicators about the result
timeNs	the buffer's timestamp in nanoseconds
timeoutUs	the timeout in microseconds

Returns

the number of elements read per buffer or error code

◆ acquireWriteBuffer()

```
virtual int SoapySDR::Device::acquireWriteBuffer ( Stream *     stream,
                                                   size_t &     handle,
                                                   void **      buffs,
                                                   const long   timeoutUs = 100000
                                                 )                                    virtual
```

Acquire direct buffers from a transmit stream. This call is part of the direct buffer access API.

The buffs array will be filled with a stream pointer for each channel. Each pointer can be written up to the number of return value elements.

The handle will be set by the implementation so that the caller may later release access to the buffers with releaseWriteBuffer(). Handle represents an index into the internal scatter/gather table such that handle is between 0 and num direct buffers - 1.

Parameters

stream	the opaque pointer to a stream handle
handle	an index value used in the release() call
buffs	an array of void* buffers num chans in size
timeoutUs	the timeout in microseconds

Returns

the number of available elements per buffer or error

◆ activateStream()

```
virtual int SoapySDR::Device::activateStream ( Stream *          stream,
                                               const int         flags = 0,
                                               const long long   timeNs = 0,
                                               const size_t      numElems = 0
                                             )
```

virtual

Activate a stream. Call activate to prepare a stream before using read/write(). The implementation control switches or stimulate data flow.

The timeNs is only valid when the flags have SOAPY_SDR_HAS_TIME. The numElems count can be used to request a finite burst size. The SOAPY_SDR_END_BURST flag can signal end on the finite burst. Not all implementations will support the full range of options. In this case, the implementation returns SOAPY_SDR_NOT_SUPPORTED.

Parameters

stream	the opaque pointer to a stream handle
flags	optional flag indicators about the stream
timeNs	optional activation time in nanoseconds
numElems	optional element count for burst control

Returns

0 for success or error code on failure

◆ closeStream()

```
virtual void SoapySDR::Device::closeStream ( Stream *  stream )
```

virtual

Close an open stream created by setupStream The implementation may change switches or power-down components.

Parameters

stream	the opaque pointer to a stream handle

◆ deactivateStream()

virtual int SoapySDR::Device::deactivateStream (Stream * stream,
const int flags = 0,
const long long timeNs = 0
) virtual

Deactivate a stream. Call deactivate when not using using read/write(). The implementation control switches or halt data flow.

The timeNs is only valid when the flags have SOAPY_SDR_HAS_TIME. Not all implementations will support the full range of options. In this case, the implementation returns SOAPY_SDR_NOT_SUPPORTED.

Parameters

- **stream** the opaque pointer to a stream handle
- **flags** optional flag indicators about the stream
- **timeNs** optional deactivation time in nanoseconds

Returns

0 for success or error code on failure

◆ enumerate() [1/2]

static KwargsList SoapySDR::Device::enumerate (const Kwargs & args = Kwargs()) static

Enumerate a list of available devices on the system.

Parameters

- **args** device construction key/value argument filters

Returns

a list of argument maps, each unique to a device

◆ enumerate() [2/2]

static KwargsList SoapySDR::Device::enumerate (const std::string & args) static

Enumerate a list of available devices on the system. Markup format for args: "keyA=valA, keyB=valB".

Parameters

args a markup string of key/value argument filters

Returns

a list of argument maps, each unique to a device

◆ getAntenna()

virtual std::string SoapySDR::Device::getAntenna (const int direction,
const size_t channel
) const virtual

Get the selected antenna on a chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the name of an available antenna

◆ getBandwidth()

virtual double SoapySDR::Device::getBandwidth (const int direction,
const size_t channel
) const virtual

Get the baseband filter width of the chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the baseband filter width in Hz

◆ getBandwidthRange()

virtual RangeList SoapySDR::Device::getBandwidthRange (const int direction,
const size_t channel
) const virtual

Get the range of possible baseband filter widths.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of bandwidth ranges in Hz

◆ getChannelInfo()

virtual Kwargs SoapySDR::Device::getChannelInfo (const int direction,
const size_t channel
) const virtual

Query a dictionary of available channel information. This dictionary can any number of values like decoder type, version, available functions... This information can be displayed to the user to help identify the instantiated channel.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

channel information

◆ getClockSource()

virtual std::string SoapySDR::Device::getClockSource (void) const virtual

Get the clock source of the device

Returns

the name of a clock source

◆ getDCOffset()

virtual std::complex<double> SoapySDR::Device::getDCOffset (const int direction,
const size_t channel
) const virtual

Get the frontend DC offset correction.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the relative correction (1.0 max)

◆ getDCOffsetMode()

virtual bool SoapySDR::Device::getDCOffsetMode (const int direction,
const size_t channel
) const virtual

Get the automatic DC offset corrections mode.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true for automatic offset correction

◆ getDirectAccessBufferAddrs()

virtual int SoapySDR::Device::getDirectAccessBufferAddrs (Stream * stream,
const size_t handle,
void ** buffs
) virtual

Get the buffer addresses for a scatter/gather table entry. When the underlying DMA implementation uses scatter/gather then this call provides the user addresses for that table.

Example: The caller may query the DMA memory addresses once after stream creation to pre-allocate a re-usable ring-buffer.

Parameters

stream the opaque pointer to a stream handle

handle an index value between 0 and num direct buffers - 1

buffs an array of void* buffers num chans in size

Returns

0 for success or error code when not supported

◆ getDriverKey()

virtual std::string SoapySDR::Device::getDriverKey (void) const virtual

A key that uniquely identifies the device driver. This key identifies the underlying implementation. Serveral variants of a product may share a driver.

◆ getFrequency() [1/2]

virtual double SoapySDR::Device::getFrequency (const int direction,
const size_t channel
) const virtual

Get the overall center frequency of the chain.

- For RX, this specifies the down-conversion frequency.
- For TX, this specifies the up-conversion frequency.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the center frequency in Hz

◆ getFrequency() [2/2]

virtual double SoapySDR::Device::getFrequency (const int direction,
const size_t channel,
const std::string & name
) const virtual

Get the frequency of a tunable element in the chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

name the name of a tunable element

Returns

the tunable element's frequency in Hz

◆ getFrequencyArgsInfo()

virtual **ArgInfoList** SoapySDR::Device::getFrequencyArgsInfo (const int direction,
const size_t channel
) const virtual

Query the argument info description for tune args.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of argument info structures

◆ getFrequencyCorrection()

virtual double SoapySDR::Device::getFrequencyCorrection (const int direction,
const size_t channel
) const virtual

Get the frontend frequency correction value.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the correction value in PPM

◆ getFrequencyRange() [1/2]

virtual RangeList SoapySDR::Device::getFrequencyRange (const int direction,
const size_t channel
) const virtual

Get the range of overall frequency values.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of frequency ranges in Hz

◆ getFrequencyRange() [2/2]

virtual RangeList SoapySDR::Device::getFrequencyRange (const int direction,
const size_t channel,
const std::string & name
) const virtual

Get the range of tunable values for the specified element.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

name the name of a tunable element

Returns

a list of frequency ranges in Hz

◆ getFrontendMapping()

virtual std::string SoapySDR::Device::getFrontendMapping (const int direction) const virtual

Get the mapping configuration string.

Parameters

direction the channel direction RX or TX

Returns

the vendor-specific mapping string

◆ getFullDuplex()

virtual bool SoapySDR::Device::getFullDuplex (const int direction,
const size_t channel
) const virtual

Find out if the specified channel is full or half duplex.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true for full duplex, false for half duplex

◆ getGain() [1/2]

virtual double SoapySDR::Device::getGain (const int direction,
const size_t channel
) const virtual

Get the overall value of the gain elements in a chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the value of the gain in dB

◆ getGain() [2/2]

virtual double SoapySDR::Device::getGain (const int direction,
const size_t channel,
const std::string & name
) const virtual

Get the value of an individual amplification element in a chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

name the name of an amplification element

Returns

the value of the gain in dB

◆ getGainMode()

virtual bool SoapySDR::Device::getGainMode (const int direction,
const size_t channel
) const virtual

Get the automatic gain mode on the chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true for automatic gain setting

◆ getGainRange() [1/2]

virtual Range SoapySDR::Device::getGainRange (const int direction,
const size_t channel
) const virtual

Get the overall range of possible gain values.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of gain ranges in dB

◆ getGainRange() [2/2]

virtual Range SoapySDR::Device::getGainRange (const int direction,
const size_t channel,
const std::string & name
) const virtual

Get the range of possible gain values for a specific element.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

name the name of an amplification element

Returns

a list of gain ranges in dB

◆ getHardwareInfo()

virtual Kwargs SoapySDR::Device::getHardwareInfo (void) const virtual

Query a dictionary of available device information. This dictionary can any number of values like vendor name, product name, revisions, serials... This information can be displayed to the user to help identify the instantiated device.

◆ getHardwareKey()

virtual std::string SoapySDR::Device::getHardwareKey (void) const virtual

A key that uniquely identifies the hardware. This key should be meaningful to the user to optimize for the underlying hardware.

◆ getHardwareTime()

virtual long long SoapySDR::Device::getHardwareTime (const std::string & what = "") const virtual

Read the time from the hardware clock on the device. The what argument can refer to a specific time counter.

Parameters

what optional argument

Returns

the time in nanoseconds

◆ getIQBalance()

virtual std::complex<double> SoapySDR::Device::getIQBalance (const int direction,
const size_t channel
) const virtual

Get the frontend IQ balance correction.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the relative correction (1.0 max)

◆ getIQBalanceMode()

virtual bool SoapySDR::Device::getIQBalanceMode (const int direction,
const size_t channel
) const virtual

Set the automatic IQ balance corrections mode.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true for automatic correction

◆ getMasterClockRate()

virtual double SoapySDR::Device::getMasterClockRate (void) const virtual

Get the master clock rate of the device.

Returns

the clock rate in Hz

◆ getMasterClockRates()

virtual RangeList SoapySDR::Device::getMasterClockRates (void) const virtual

Get the range of available master clock rates.

Returns

a list of clock rate ranges in Hz

◆ getNativeDeviceHandle()

virtual void* SoapySDR::Device::getNativeDeviceHandle (void) const virtual

A handle to the native device used by the driver. The implementation may return a null value if it does not support or does not wish to provide access to the native handle.

Returns

a handle to the native device or null

◆ getNativeStreamFormat()

virtual std::string SoapySDR::Device::getNativeStreamFormat (const int direction, const size_t channel, double & fullScale) const virtual

Get the hardware's native stream format for this channel. This is the format used by the underlying transport layer, and the direct buffer access API calls (when available).

Parameters

	direction	the channel direction RX or TX
	channel	an available channel on the device
`[out]`	**fullScale**	the maximum possible value

Returns

the native stream buffer format string

◆ getNumChannels()

virtual size_t SoapySDR::Device::getNumChannels (const int direction) const virtual

Get a number of channels given the streaming direction

◆ getNumDirectAccessBuffers()

virtual size_t SoapySDR::Device::getNumDirectAccessBuffers (Stream * stream) virtual

How many direct access buffers can the stream provide? This is the number of times the user can call acquire() on a stream without making subsequent calls to release(). A return value of 0 means that direct access is not supported.

Parameters

stream the opaque pointer to a stream handle

Returns

the number of direct access buffers or 0

◆ getReferenceClockRate()

virtual double SoapySDR::Device::getReferenceClockRate (void) const virtual

Get the reference clock rate of the device.

Returns

the clock rate in Hz

◆ getReferenceClockRates()

virtual RangeList SoapySDR::Device::getReferenceClockRates (void) const virtual

Get the range of available reference clock rates.

Returns

a list of clock rate ranges in Hz

◆ getSampleRate()

virtual double SoapySDR::Device::getSampleRate (const int direction,
const size_t channel
) const virtual

Get the baseband sample rate of the chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

the sample rate in samples per second

◆ getSampleRateRange()

virtual RangeList SoapySDR::Device::getSampleRateRange (const int direction,
const size_t channel
) const virtual

Get the range of possible baseband sample rates.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of sample rate ranges in samples per second

◆ getSensorInfo() [1/2]

virtual ArgInfo SoapySDR::Device::getSensorInfo (const int direction,
const size_t channel,
const std::string & key
) const virtual

Get meta-information about a channel sensor. Example: displayable name, type, range.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

key the ID name of an available sensor

Returns

meta-information about a sensor

◆ getSensorInfo() [2/2]

virtual ArgInfo SoapySDR::Device::getSensorInfo (const std::string & key) const virtual

Get meta-information about a sensor. Example: displayable name, type, range.

Parameters

key the ID name of an available sensor

Returns

meta-information about a sensor

◆ getSettingInfo() [1/2]

virtual ArgInfoList SoapySDR::Device::getSettingInfo (const int direction,
const size_t channel
) const virtual

Describe the allowed keys and values used for channel settings.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of argument info structures

◆ getSettingInfo() [2/2]

virtual ArgInfoList SoapySDR::Device::getSettingInfo (void) const virtual

Describe the allowed keys and values used for settings.

Returns

a list of argument info structures

◆ getStreamArgsInfo()

virtual ArgInfoList SoapySDR::Device::getStreamArgsInfo (const int direction,
const size_t channel
) const virtual

Query the argument info description for stream args.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of argument info structures

◆ getStreamFormats()

virtual std::vector<std::string> SoapySDR::Device::getStreamFormats (const int direction,
const size_t channel
) const virtual

Query a list of the available stream formats.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of allowed format strings. See setupStream() for the format syntax.

◆ getStreamMTU()

virtual size_t SoapySDR::Device::getStreamMTU (Stream * stream) const virtual

Get the stream's maximum transmission unit (MTU) in number of elements. The MTU specifies the maximum payload transfer in a stream operation. This value can be used as a stream buffer allocation size that can best optimize throughput given the underlying stream implementation.

Parameters

stream the opaque pointer to a stream handle

Returns

the MTU in number of stream elements (never zero)

◆ getTimeSource()

virtual std::string SoapySDR::Device::getTimeSource (void) const virtual

Get the time source of the device

Returns

the name of a time source

◆ hasDCOffset()

virtual bool SoapySDR::Device::hasDCOffset (const int direction,
const size_t channel
) const virtual

Does the device support frontend DC offset correction?

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true if DC offset corrections are supported

◆ hasDCOffsetMode()

virtual bool SoapySDR::Device::hasDCOffsetMode (const int direction,
const size_t channel
) const virtual

Does the device support automatic DC offset corrections?

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true if automatic corrections are supported

◆ hasFrequencyCorrection()

virtual bool SoapySDR::Device::hasFrequencyCorrection (const int direction,
const size_t channel
) const virtual

Does the device support frontend frequency correction?

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true if frequency corrections are supported

◆ hasGainMode()

virtual bool SoapySDR::Device::hasGainMode (const int direction,
const size_t channel
) const virtual

Does the device support automatic gain control?

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true for automatic gain control

◆ hasHardwareTime()

virtual bool SoapySDR::Device::hasHardwareTime (const std::string & what = "") const virtual

Does this device have a hardware clock?

Parameters

what optional argument

Returns

true if the hardware clock exists

◆ hasIQBalance()

virtual bool SoapySDR::Device::hasIQBalance (const int direction,
const size_t channel
) const virtual

Does the device support frontend IQ balance correction?

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true if IQ balance corrections are supported

◆ hasIQBalanceMode()

virtual bool SoapySDR::Device::hasIQBalanceMode (const int direction,
const size_t channel
) const virtual

Does the device support automatic frontend IQ balance correction?

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

true if IQ balance corrections are supported

◆ listAntennas()

virtual std::vector<std::string> SoapySDR::Device::listAntennas (const int direction,
const size_t channel
) const virtual

Get a list of available antennas to select on a given chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of available antenna names

◆ listBandwidths()

virtual std::vector<double> SoapySDR::Device::listBandwidths (const int direction,
const size_t channel
) const virtual

Get the range of possible baseband filter widths.

Deprecated:

replaced by getBandwidthRange()

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of possible bandwidths in Hz

◆ listClockSources()

virtual std::vector<std::string> SoapySDR::Device::listClockSources (void) const virtual

Get the list of available clock sources.

Returns

a list of clock source names

◆ listFrequencies()

virtual std::vector<std::string> SoapySDR::Device::listFrequencies (const int direction,
const size_t channel
) const virtual

List available tunable elements in the chain. Elements should be in order RF to baseband.

Parameters

direction the channel direction RX or TX

channel an available channel

Returns

a list of tunable elements by name

◆ listGains()

virtual std::vector<std::string> SoapySDR::Device::listGains (const int direction,
const size_t channel
) const virtual

List available amplification elements. Elements should be in order RF to baseband.

Parameters

direction the channel direction RX or TX

channel an available channel

Returns

a list of gain string names

◆ listGPIOBanks()

virtual std::vector<std::string> SoapySDR::Device::listGPIOBanks (void) const virtual

Get a list of available GPIO banks by name.

◆ listRegisterInterfaces()

virtual std::vector<std::string> SoapySDR::Device::listRegisterInterfaces (void) const virtual

Get a list of available register interfaces by name.

Returns

a list of available register interfaces

◆ listSampleRates()

virtual std::vector<double> SoapySDR::Device::listSampleRates (const int direction, const size_t channel) const virtual

Get the range of possible baseband sample rates.

Deprecated:

replaced by getSampleRateRange()

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of possible rates in samples per second

◆ listSensors() [1/2]

virtual std::vector<std::string> SoapySDR::Device::listSensors (const int direction, const size_t channel) const virtual

List the available channel readback sensors. A sensor can represent a reference lock, RSSI, temperature.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

Returns

a list of available sensor string names

◆ listSensors() [2/2]

virtual std::vector<std::string> SoapySDR::Device::listSensors (void) const virtual

List the available global readback sensors. A sensor can represent a reference lock, RSSI, temperature.

Returns

a list of available sensor string names

◆ listTimeSources()

virtual std::vector<std::string> SoapySDR::Device::listTimeSources (void) const virtual

Get the list of available time sources.

Returns

a list of time source names

◆ listUARTs()

virtual std::vector<std::string> SoapySDR::Device::listUARTs (void) const virtual

Enumerate the available UART devices.

Returns

a list of names of available UARTs

◆ make() [1/4]

static Device* SoapySDR::Device::make (const Kwargs & args = Kwargs()) static

Make a new Device object given device construction args. The device pointer will be stored in a table so subsequent calls with the same arguments will produce the same device. For every call to make, there should be a matched call to unmake.

Parameters

args device construction key/value argument map

Returns

a pointer to a new Device object

◆ make() [2/4]

static std::vector<Device *> SoapySDR::Device::make (const KwargsList & argsList) static

Create a list of devices from a list of construction arguments. This is a convenience call to parallelize device construction, and is fundamentally a parallel for loop of make(Kwargs).

Parameters

argsList a list of device arguments per each device

Returns

a list of device pointers per each specified argument

◆ make() [3/4]

static Device* SoapySDR::Device::make (const std::string & args) static

Make a new Device object given device construction args. The device pointer will be stored in a table so subsequent calls with the same arguments will produce the same device. For every call to make, there should be a matched call to unmake.

Parameters

args a markup string of key/value arguments

Returns

a pointer to a new Device object

◆ make() [4/4]

static std::vector<Device *> SoapySDR::Device::make (const std::vector< std::string > & argsList) static

Create a list of devices from a list of construction arguments. This is a convenience call to parallelize device construction, and is fundamentally a parallel for loop of make(args).

Parameters

argsList a list of device arguments per each device

Returns

a list of device pointers per each specified argument

◆ readGPIO()

virtual unsigned SoapySDR::Device::readGPIO (const std::string & bank) const virtual

Readback the value of a GPIO bank.

Parameters

bank the name of an available bank

Returns

an integer representing GPIO bits

◆ readGPIODir()

virtual unsigned SoapySDR::Device::readGPIODir (const std::string & bank) const virtual

Read the data direction of a GPIO bank. 1 bits represent outputs, 0 bits represent inputs.

Parameters

bank the name of an available bank

Returns

an integer representing data direction bits

◆ readI2C()

virtual std::string SoapySDR::Device::readI2C (const int addr,
const size_t numBytes
) virtual

Read from an available I2C slave. If the device contains multiple I2C masters, the address bits can encode which master.

Parameters

addr the address of the slave

numBytes the number of bytes to read

Returns

an array of bytes read from the slave

◆ readRegister() [1/2]

virtual unsigned SoapySDR::Device::readRegister (const std::string & name,
const unsigned addr
) const virtual

Read a register on the device given the interface name.

Parameters

name the name of a available register interface

addr the register address

Returns

the register value

◆ readRegister() [2/2]

virtual unsigned SoapySDR::Device::readRegister (const unsigned addr) const virtual

Read a register on the device.

Deprecated:
replaced by readRegister(name)

Parameters

addr the register address

Returns

the register value

◆ readRegisters()

virtual std::vector<unsigned> SoapySDR::Device::readRegisters (const std::string & name,
const unsigned addr,
const size_t length
) const virtual

Read a memory block on the device given the interface name.

Parameters

name the name of a available memory block interface

addr the memory block start address

length number of words to be read from memory block

Returns

the memory block content

◆ readSensor() [1/4]

```
virtual std::string SoapySDR::Device::readSensor ( const int direction,
                                                   const size_t channel,
                                                   const std::string & key
                                                 ) const
```

virtual

Readback a channel sensor given the name. The value returned is a string which can represent a boolean ("true"/"false"), an integer, or float.

Parameters

- **direction** the channel direction RX or TX
- **channel** an available channel on the device
- **key** the ID name of an available sensor

Returns

the current value of the sensor

◆ readSensor() [2/4]

```
template<typename Type >
Type SoapySDR::Device::readSensor ( const int direction,
                                    const size_t channel,
                                    const std::string & key
                                  ) const
```

Readback a channel sensor given the name.

Template Parameters

- **Type** the return type for the sensor value

Parameters

- **direction** the channel direction RX or TX
- **channel** an available channel on the device
- **key** the ID name of an available sensor

Returns

the current value of the sensor as the specified type

◆ readSensor() [3/4]

virtual std::string SoapySDR::Device::readSensor (const std::string & key) const virtual

Readback a global sensor given the name. The value returned is a string which can represent a boolean ("true"/"false"), an integer, or float.

Parameters

key the ID name of an available sensor

Returns

the current value of the sensor

◆ readSensor() [4/4]

template<typename Type >

Type SoapySDR::Device::readSensor (const std::string & key) const

Readback a global sensor given the name.

Template Parameters

Type the return type for the sensor value

Parameters

key the ID name of an available sensor

Returns

the current value of the sensor as the specified type

◆ readSetting() [1/4]

```
template<typename Type >
Type SoapySDR::Device::readSetting ( const int          direction,
                                     const size_t       channel,
                                     const std::string & key
                                   )
```

Read an arbitrary channel setting on the device.

Template Parameters

Type the return type for the sensor value

Parameters

direction the channel direction RX or TX

channel an available channel on the device

key the setting identifier

Returns

the setting value

◆ readSetting() [2/4]

```
virtual std::string SoapySDR::Device::readSetting ( const int          direction,
                                                    const size_t       channel,
                                                    const std::string & key
                                                  ) const                          virtual
```

Read an arbitrary channel setting on the device.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

key the setting identifier

Returns

the setting value

◆ readSetting() [3/4]

template<typename Type >

Type SoapySDR::Device::readSetting (const std::string & key)

Read an arbitrary setting on the device.

Template Parameters

Type the return type for the sensor value

Parameters

key the setting identifier

Returns

the setting value

◆ readSetting() [4/4]

virtual std::string SoapySDR::Device::readSetting (const std::string & key) const virtual

Read an arbitrary setting on the device.

Parameters

key the setting identifier

Returns

the setting value

◆ readStream()

```
virtual int SoapySDR::Device::readStream ( Stream *        stream,
                                           void *const *   buffs,
                                           const size_t    numElems,
                                           int &           flags,
                                           long long &     timeNs,
                                           const long      timeoutUs = 100000
                                         )
```

virtual

Read elements from a stream for reception. This is a multi-channel call, and buffs should be an array of void *, where each pointer will be filled with data from a different channel.

Client code compatibility: The readStream() call should be well defined at all times, including prior to activation and after deactivation. When inactive, readStream() should implement the timeout specified by the caller and return SOAPY_SDR_TIMEOUT.

Parameters

stream	the opaque pointer to a stream handle
buffs	an array of void* buffers num chans in size
numElems	the number of elements in each buffer
flags	optional flag indicators about the result
timeNs	the buffer's timestamp in nanoseconds
timeoutUs	the timeout in microseconds

Returns

the number of elements read per buffer or error code

◆ readStreamStatus()

```
virtual int SoapySDR::Device::readStreamStatus ( Stream *      stream,
                                                 size_t &      chanMask,
                                                 int &         flags,
                                                 long long &   timeNs,
                                                 const long    timeoutUs = 100000
                                               )                                    virtual
```

Readback status information about a stream. This call is typically used on a transmit stream to report time errors, underflows, and burst completion.

Client code compatibility: Client code may continually poll readStreamStatus() in a loop. Implementations of readStreamStatus() should wait in the call for a status change event or until the timeout expiration. When stream status is not implemented on a particular stream, readStreamStatus() should return SOAPY_SDR_NOT_SUPPORTED. Client code may use this indication to disable a polling loop.

Parameters

stream	the opaque pointer to a stream handle
chanMask	to which channels this status applies
flags	optional input flags and output flags
timeNs	the buffer's timestamp in nanoseconds
timeoutUs	the timeout in microseconds

Returns

0 for success or error code like timeout

◆ readUART()

```
virtual std::string SoapySDR::Device::readUART ( const std::string &  which,
                                                 const long           timeoutUs = 100000
                                                 const
                                               )                                    virtual
```

Read bytes from a UART until timeout or newline. Its up to the implementation to set the baud rate, carriage return settings, flushing on newline.

Parameters

which	the name of an available UART
timeoutUs	a timeout in microseconds

Returns

an array of bytes read from the UART

◆ releaseReadBuffer()

```
virtual void SoapySDR::Device::releaseReadBuffer ( Stream *       stream,
                                                   const size_t   handle
                                                 )                                  virtual
```

Release an acquired buffer back to the receive stream. This call is part of the direct buffer access API.

Parameters

stream	the opaque pointer to a stream handle
handle	the opaque handle from the acquire() call

◆ releaseWriteBuffer()

```
virtual void SoapySDR::Device::releaseWriteBuffer ( Stream *          stream,
                                                    const size_t      handle,
                                                    const size_t      numElems,
                                                    int &             flags,
                                                    const long long   timeNs = 0
                                                  )                                 virtual
```

Release an acquired buffer back to the transmit stream. This call is part of the direct buffer access API.

Stream meta-data is provided as part of the release call, and not the acquire call so that the caller may acquire buffers without committing to the contents of the meta-data, which can be determined by the user as the buffers are filled.

Parameters

stream	the opaque pointer to a stream handle
handle	the opaque handle from the acquire() call
numElems	the number of elements written to each buffer
flags	optional input flags and output flags
timeNs	the buffer's timestamp in nanoseconds

◆ setAntenna()

```
virtual void SoapySDR::Device::setAntenna ( const int          direction,
                                            const size_t       channel,
                                            const std::string & name
                                          )                                virtual
```

Set the selected antenna on a chain.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
name	the name of an available antenna

◆ setBandwidth()

```
virtual void SoapySDR::Device::setBandwidth ( const int    direction,
                                              const size_t channel,
                                              const double bw
                                            )                              virtual
```

Set the baseband filter width of the chain.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
bw	the baseband filter width in Hz

◆ setClockSource()

```
virtual void SoapySDR::Device::setClockSource ( const std::string & source )   virtual
```

Set the clock source on the device

Parameters

source	the name of a clock source

◆ setCommandTime()

```
virtual void SoapySDR::Device::setCommandTime ( const long long      timeNs,
                                                const std::string &  what = " "
                                              )                                  virtual
```

Set the time of subsequent configuration calls. The what argument can refer to a specific command queue. Implementations may use a time of 0 to clear.

Deprecated:
replaced by setHardwareTime()

Parameters

timeNs	time in nanoseconds
what	optional argument

◆ setDCOffset()

```
virtual void SoapySDR::Device::setDCOffset ( const int                          direction,
                                             const size_t                       channel,
                                             const std::complex< double > &     offset
                                           )                                    virtual
```

Set the frontend DC offset correction.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
offset	the relative correction (1.0 max)

◆ setDCOffsetMode()

virtual void SoapySDR::Device::setDCOffsetMode (const int direction,
const size_t channel,
const bool automatic
)

virtual

Set the automatic DC offset corrections mode.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
automatic	true for automatic offset correction

◆ setFrequency() [1/2]

```
virtual void SoapySDR::Device::setFrequency ( const int          direction,
                                              const size_t       channel,
                                              const double       frequency,
                                              const Kwargs &     args = Kwargs ( )
                                            )
```

virtual

Set the center frequency of the chain.

- For RX, this specifies the down-conversion frequency.
- For TX, this specifies the up-conversion frequency.

The default implementation of setFrequency() will tune the "RF" component as close as possible to the requested center frequency. Tuning inaccuracies will be compensated for with the "BB" component.

The args can be used to augment the tuning algorithm.

- Use "OFFSET" to specify an "RF" tuning offset, usually with the intention of moving the LO out of the passband. The offset will be compensated for using the "BB" component.
- Use the name of a component for the key and a frequency in Hz as the value (any format) to enforce a specific frequency. The other components will be tuned with compensation to achieve the specified overall frequency.
- Use the name of a component for the key and the value "IGNORE" so that the tuning algorithm will avoid altering the component.
- Vendor specific implementations can also use the same args to augment tuning in other ways such as specifying fractional vs integer N tuning.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
frequency	the center frequency in Hz
args	optional tuner arguments

◆ setFrequency() [2/2]

```
virtual void SoapySDR::Device::setFrequency ( const int          direction,
                                              const size_t       channel,
                                              const std::string & name,
                                              const double       frequency,
                                              const Kwargs &     args = Kwargs ( )
                                            )                                    virtual
```

Tune the center frequency of the specified element.

- For RX, this specifies the down-conversion frequency.
- For TX, this specifies the up-conversion frequency.

Recommended names used to represent tunable components:

- "CORR" - freq error correction in PPM
- "RF" - frequency of the RF frontend
- "BB" - frequency of the baseband DSP

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
name	the name of a tunable element
frequency	the center frequency in Hz
args	optional tuner arguments

◆ setFrequencyCorrection()

```
virtual void SoapySDR::Device::setFrequencyCorrection ( const int    direction,
                                                        const size_t channel,
                                                        const double value
                                                      )                          virtual
```

Fine tune the frontend frequency correction.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
value	the correction in PPM

◆ setFrontendMapping()

virtual void SoapySDR::Device::setFrontendMapping (const int direction,
const std::string & mapping
) virtual

Set the frontend mapping of available DSP units to RF frontends. This mapping controls channel mapping and channel availability.

Parameters

direction the channel direction RX or TX

mapping a vendor-specific mapping string

◆ setGain() [1/2]

virtual void SoapySDR::Device::setGain (const int direction,
const size_t channel,
const double value
) virtual

Set the overall amplification in a chain. The gain will be distributed automatically across available element.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

value the new amplification value in dB

◆ setGain() [2/2]

```
virtual void SoapySDR::Device::setGain ( const int          direction,
                                         const size_t       channel,
                                         const std::string & name,
                                         const double       value
                                       )                                 virtual
```

Set the value of a amplification element in a chain.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
name	the name of an amplification element
value	the new amplification value in dB

◆ setGainMode()

```
virtual void SoapySDR::Device::setGainMode ( const int      direction,
                                             const size_t    channel,
                                             const bool      automatic
                                           )                             virtual
```

Set the automatic gain mode on the chain.

Parameters

direction	the channel direction RX or TX
channel	an available channel on the device
automatic	true for automatic gain setting

◆ setHardwareTime()

```
virtual void SoapySDR::Device::setHardwareTime ( const long long      timeNs,
                                                 const std::string &  what = " "
                                               )                                  virtual
```

Write the time to the hardware clock on the device. The what argument can refer to a specific time counter.

Parameters

timeNs time in nanoseconds

what optional argument

◆ setIQBalance()

```
virtual void SoapySDR::Device::setIQBalance ( const int                          direction,
                                              const size_t                       channel,
                                              const std::complex< double > &     balance
                                            )                                    virtual
```

Set the frontend IQ balance correction.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

balance the relative correction (1.0 max)

◆ setIQBalanceMode()

```
virtual void SoapySDR::Device::setIQBalanceMode ( const int      direction,
                                                  const size_t   channel,
                                                  const bool     automatic
                                                )                            virtual
```

Set the automatic frontend IQ balance correction.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

automatic true for automatic correction

◆ setMasterClockRate()

virtual void SoapySDR::Device::setMasterClockRate (const double rate) virtual

Set the master clock rate of the device.

Parameters

rate the clock rate in Hz

◆ setReferenceClockRate()

virtual void SoapySDR::Device::setReferenceClockRate (const double rate) virtual

Set the reference clock rate of the device.

Parameters

rate the clock rate in Hz

◆ setSampleRate()

virtual void SoapySDR::Device::setSampleRate (const int direction,
const size_t channel,
const double rate
) virtual

Set the baseband sample rate of the chain.

Parameters

direction the channel direction RX or TX

channel an available channel on the device

rate the sample rate in samples per second

◆ setTimeSource()

virtual void SoapySDR::Device::setTimeSource (const std::string & source) virtual

Set the time source on the device

Parameters

source the name of a time source

◆ setupStream()

```
virtual Stream*
SoapySDR::Device::setupStream ( const int                    direction,
                                const std::string &          format,
                                const std::vector< size_t > & channels = std::vector< size_t >(),
                                const Kwargs &               args = Kwargs()
                              )                                                              virtual
```

Initialize a stream given a list of channels and stream arguments. The implementation may change switches or power-up components. All stream API calls should be usable with the new stream object after setupStream() is complete, regardless of the activity state.

The API allows any number of simultaneous TX and RX streams, but many dual-channel devices are limited to one stream in each direction, using either one or both channels. This call will throw an exception if an unsupported combination is requested, or if a requested channel in this direction is already in use by another stream.

When multiple channels are added to a stream, they are typically expected to have the same sample rate. See setSampleRate().

Parameters

direction the channel direction (SOAPY_SDR_RX or SOAPY_SDR_TX)

format A string representing the desired buffer format in read/writeStream()

The first character selects the number type:

- "C" means complex
- "F" means floating point
- "S" means signed integer
- "U" means unsigned integer

The type character is followed by the number of bits per number (complex is 2x this size per sample)

Example format strings:

- "CF32" - complex float32 (8 bytes per element)
- "CS16" - complex int16 (4 bytes per element)
- "CS12" - complex int12 (3 bytes per element)
- "CS4" - complex int4 (1 byte per element)
- "S32" - int32 (4 bytes per element)
- "U8" - uint8 (1 byte per element)

channels a list of channels or empty for automatic.

args stream args or empty for defaults.

Recommended keys to use in the args dictionary:

- "WIRE" - format of the samples between device and host

Returns

an opaque pointer to a stream handle.

The returned stream is not required to have internal locking, and may not be used concurrently from multiple threads.

◆ transactSPI()

virtual unsigned SoapySDR::Device::transactSPI (const int addr,
const unsigned data,
const size_t numBits
) virtual

Perform a SPI transaction and return the result. Its up to the implementation to set the clock rate, and read edge, and the write edge of the SPI core. SPI slaves without a readback pin will return 0.

If the device contains multiple SPI masters, the address bits can encode which master.

Parameters

addr	an address of an available SPI slave
data	the SPI data, numBits-1 is first out
numBits	the number of bits to clock out

Returns

the readback data, numBits-1 is first in

◆ unmake() [1/2]

static void SoapySDR::Device::unmake (const std::vector< Device * > & devices) static

Unmake or release a list of device handles. This is a convenience call to parallelize device destruction, and is fundamentally a parallel for loop of unmake(Device *).

Parameters

devices a list of pointers to device objects

◆ unmake() [2/2]

static void SoapySDR::Device::unmake (Device * device) static

Unmake or release a device object handle.

Parameters

device a pointer to a device object

◆ writeGPIO() [1/2]

virtual void SoapySDR::Device::writeGPIO (const std::string & bank,
const unsigned value
) virtual

Write the value of a GPIO bank.

Parameters

bank the name of an available bank

value an integer representing GPIO bits

◆ writeGPIO() [2/2]

virtual void SoapySDR::Device::writeGPIO (const std::string & bank,
const unsigned value,
const unsigned mask
) virtual

Write the value of a GPIO bank with modification mask.

Parameters

bank the name of an available bank

value an integer representing GPIO bits

mask a modification mask where 1 = modify

◆ writeGPIODir() [1/2]

virtual void SoapySDR::Device::writeGPIODir (const std::string & bank,
const unsigned dir
) virtual

Write the data direction of a GPIO bank. 1 bits represent outputs, 0 bits represent inputs.

Parameters

bank the name of an available bank

dir an integer representing data direction bits

◆ writeGPIODir() [2/2]

virtual void SoapySDR::Device::writeGPIODir (const std::string & bank,
const unsigned dir,
const unsigned mask
) virtual

Write the data direction of a GPIO bank with modification mask. 1 bits represent outputs, 0 bits represent inputs.

Parameters

bank the name of an available bank

dir an integer representing data direction bits

mask a modification mask where 1 = modify

◆ writeI2C()

virtual void SoapySDR::Device::writeI2C (const int addr,
const std::string & data
) virtual

Write to an available I2C slave. If the device contains multiple I2C masters, the address bits can encode which master.

Parameters

addr the address of the slave

data an array of bytes write out

◆ writeRegister() [1/2]

virtual void SoapySDR::Device::writeRegister (const std::string & name,
const unsigned addr,
const unsigned value
) virtual

Write a register on the device given the interface name. This can represent a register on a soft CPU, FPGA, IC; the interpretation is up the implementation to decide.

Parameters

name the name of a available register interface

addr the register address

value the register value

◆ writeRegister() [2/2]

virtual void SoapySDR::Device::writeRegister (const unsigned addr,
const unsigned value
) virtual

Write a register on the device. This can represent a register on a soft CPU, FPGA, IC; the interpretation is up the implementation to decide.

Deprecated:
replaced by writeRegister(name)

Parameters

addr the register address

value the register value

◆ writeRegisters()

```
virtual void SoapySDR::Device::writeRegisters ( const std::string &                name,
                                                const unsigned                     addr,
                                                const std::vector< unsigned > &    value
                                              )                                                virtual
```

Write a memory block on the device given the interface name. This can represent a memory block on a soft CPU, FPGA, IC; the interpretation is up the implementation to decide.

Parameters

- **name** the name of a available memory block interface
- **addr** the memory block start address
- **value** the memory block content

◆ writeSetting() [1/4]

```
virtual void SoapySDR::Device::writeSetting ( const int             direction,
                                              const size_t          channel,
                                              const std::string &   key,
                                              const std::string &   value
                                            )                                                  virtual
```

Write an arbitrary channel setting on the device. The interpretation is up the implementation.

Parameters

- **direction** the channel direction RX or TX
- **channel** an available channel on the device
- **key** the setting identifier
- **value** the setting value

◆ writeSetting() [2/4]

```
template<typename Type >
void SoapySDR::Device::writeSetting ( const int          direction,
                                      const size_t       channel,
                                      const std::string & key,
                                      const Type &       value
                                    )
```

Write an arbitrary channel setting on the device.

Template Parameters

Type the data type of the value

Parameters

direction the channel direction RX or TX

channel an available channel on the device

key the setting identifier

value the setting value

◆ writeSetting() [3/4]

```
virtual void SoapySDR::Device::writeSetting ( const std::string & key,
                                              const std::string & value
                                            )
```

virtual

Write an arbitrary setting on the device. The interpretation is up the implementation.

Parameters

key the setting identifier

value the setting value

◆ writeSetting() [4/4]

template<typename Type >

void SoapySDR::Device::writeSetting (const std::string & key,
const Type & value
)

Write a setting with an arbitrary value type.

Template Parameters

Type the data type of the value

Parameters

key the setting identifier

value the setting value

◆ writeStream()

```
virtual int SoapySDR::Device::writeStream ( Stream *              stream,
                                            const void *const *   buffs,
                                            const size_t          numElems,
                                            int &                 flags,
                                            const long long       timeNs = 0,
                                            const long            timeoutUs = 100000
                                          )                                          virtual
```

Write elements to a stream for transmission. This is a multi-channel call, and buffs should be an array of void *, where each pointer will be filled with data for a different channel.

Client code compatibility: Client code relies on writeStream() for proper back-pressure. The writeStream() implementation must enforce the timeout such that the call blocks until space becomes available or timeout expiration.

Parameters

stream	the opaque pointer to a stream handle
buffs	an array of void* buffers num chans in size
numElems	the number of elements in each buffer
flags	optional input flags and output flags
timeNs	the buffer's timestamp in nanoseconds
timeoutUs	the timeout in microseconds

Returns

the number of elements written per buffer or error

◆ writeUART()

```
virtual void SoapySDR::Device::writeUART ( const std::string &  which,
                                           const std::string &  data
                                         )                                           virtual
```

Write data to a UART device. Its up to the implementation to set the baud rate, carriage return settings, flushing on newline.

Parameters

which	the name of an available UART
data	an array of bytes to write out

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

- Device.hpp