Other Code

Generated by Doxygen 1.9.5

Chapter 1

Other

This sub-project contains other sub-projects which support the Raspberry Pi project

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all 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 files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 behaviour_tree Namespace Reference

Namespaces

- namespace global
- namespace node

Classes

- class BehaviourTree
- class BehaviourTreeParser
- class Context
- class Root

Enumerations

- enum class Status { Success , Failure , Running }

6.1.1 Enumeration Type Documentation

6.1.1.1 Status

```
enum class behaviour_tree::Status  [strong]
```

Enumerator

Success	
Failure	
Running	

6.2 behaviour_tree::global Namespace Reference

Variables

- constexpr double CM_TO_DISTANCE = 15.151515151515151515151515151515151515151515151515151515

6.2.1 Variable Documentation

6.2.1.1 CM_TO_DISTANCE

```
constexpr double behaviour_tree::global::CM_TO_DISTANCE = 15.1515151515151515151515151515151515151515151515151
[constexpr]
```

6.3 behaviour_tree::node Namespace Reference

Namespaces

- namespace blackboard
- namespace composite
- namespace custom
- namespace decorator
- namespace leaf

Classes

- class Node

6.4 behaviour_tree::node::blackboard Namespace Reference

Classes

- class BlackboardNode
- class ChangeInteger
- class IntegerCondition

Enumerations

- enum class BlackboardType { ChangeInteger , IntegerCondition }
- enum class ConditionOperatorType {
  Equal , NotEqual , GreaterThan , GreaterThanOrEqual ,
  LessThan , LessThanOrEqual }
- enum class IntegerChangeType { Set , Add , Subtract }

6.4.1 Enumeration Type Documentation

6.4.1.1 BlackboardType

```
enum class behaviour_tree::node::blackboard::BlackboardType  [strong]
```

Enumerator

ChangeInteger	
IntegerCondition	

6.4.1.2 ConditionOperatorType

```
enum class behaviour_tree::node::blackboard::ConditionOperatorType  [strong]
```

Enumerator

Equal	
NotEqual	
GreaterThan	
GreaterThanOrEqual	
LessThan	
LessThanOrEqual	

6.4.1.3 IntegerChangeType

```
enum class behaviour_tree::node::blackboard::IntegerChangeType  [strong]
```

Enumerator

Set	
Add	
Subtract	

6.5 behaviour_tree::node::composite Namespace Reference

Classes

- class Composite
- class Random
- class Selector
- class Sequence

Enumerations

- enum class CompositeType { Sequence , Selector , Random }

6.5.1 Enumeration Type Documentation

6.5.1.1 CompositeType

```
enum class behaviour_tree::node::composite::CompositeType  [strong]
```

Enumerator

Sequence	
Selector	
Random	

6.6 behaviour_tree::node::custom Namespace Reference

Namespaces

- namespace action
- namespace condition

Classes

- class CarCustomNodeParser
- class CustomNode
- class CustomNodeParser

6.7 behaviour_tree::node::custom::action Namespace Reference

Classes

- class Drive
- class PauseExecution
- class Print
- class SetAngle
- class SetSpeed
- class SetWheelDirection
- class Turn

Enumerations

- enum class ClockDirectionType { Clockwise , AntiClockwise }
- enum class DirectionType { Forward , Backward }
- enum class ServoType { FrontWheels , CameraServo1 , CameraServo2 }
- enum class WheelType { Left , Right , Both }

6.7.1 Enumeration Type Documentation

6.7.1.1 ClockDirectionType

```
enum class behaviour_tree::node::custom::action::ClockDirectionType  [strong]
```

Enumerator

Clockwise	
AntiClockwise	

6.7.1.2 DirectionType

```
enum class behaviour_tree::node::custom::action::DirectionType  [strong]
```

Enumerator

Forward	
Backward	

6.7.1.3 ServoType

```
enum class behaviour_tree::node::custom::action::ServoType  [strong]
```

Enumerator

FrontWheels	
CameraServo1	
CameraServo2	

6.7.1.4 WheelType

```
enum class behaviour_tree::node::custom::action::WheelType  [strong]
```

Enumerator

Left	
Right	
Both	

6.8 behaviour_tree::node::custom::condition Namespace Reference

Classes

- class SucceedOnAnyNearbyScan
- class SucceedOnAverageColour
- class SucceedOnAverageNearbyScan

6.9 behaviour_tree::node::decorator Namespace Reference

Classes

- class Decorator
- class Invert
- class Repeat

Enumerations

- enum class DecoratorType { Invert , Repeat }

6.9.1 Enumeration Type Documentation

6.9.1.1 DecoratorType

```
enum class behaviour_tree::node::decorator::DecoratorType  [strong]
```

Enumerator

Invert	
Repeat	

6.10 behaviour_tree::node::leaf Namespace Reference

Classes

- class Fail
- class Leaf
- class Succeed
- class UseRoot

Enumerations

- enum class LeafType { Succeed , Fail , UseRoot }

6.10.1 Enumeration Type Documentation

6.10.1.1 LeafType

```
enum class behaviour_tree::node::leaf::LeafType  [strong]
```

Enumerator

Succeed	
Fail	
UseRoot	

6.11 rplidar Namespace Reference

Classes

- struct DeviceInfo
- class ExpressPacket
- struct HealthInfo

  Health Info for Lidar Scanner.
- struct Measure
- class RPLidar

  Class for communicating with RPLidar rangefinder scanners.
- struct ScanInfo

Enumerations

- enum ScanType { NORMAL = 0 , FORCE , EXPRESS }

Variables

- constexpr uint8_t SYNC_BYTE = 0xA5
- constexpr uint8_t SYNC_BYTE2 = 0x5A
- constexpr uint8_t GET_INFO_BYTE = 0x50
- constexpr uint8_t GET_HEALTH_BYTE = 0x52
- constexpr uint8_t STOP_BYTE = 0x25
- constexpr uint8_t RESET_BYTE = 0x40
- constexpr int DESCRIPTOR_LEN = 7
- constexpr int INFO_LEN = 20
- constexpr int HEALTH_LEN = 3
- constexpr int INFO_TYPE = 4
- constexpr int HEALTH_TYPE = 6
- constexpr int MAX_MOTOR_PWM = 1023
- constexpr int DEFAULT_MOTOR_PWM = 660
- constexpr uint8_t SET_PWM_BYTE = 0xF0

6.11.1 Enumeration Type Documentation

6.11.1.1 ScanType

```
enum rplidar::ScanType
```

Enumerator

NORMAL	
FORCE	
EXPRESS	

6.11.2 Variable Documentation

6.11.2.1 DEFAULT_MOTOR_PWM

```
constexpr int rplidar::DEFAULT_MOTOR_PWM = 660  [constexpr]
```

6.11.2.2 DESCRIPTOR_LEN

```
constexpr int rplidar::DESCRIPTOR_LEN = 7  [constexpr]
```

6.11.2.3 GET_HEALTH_BYTE

```
constexpr uint8_t rplidar::GET_HEALTH_BYTE = 0x52  [constexpr]
```

6.11.2.4 GET_INFO_BYTE

```
constexpr uint8_t rplidar::GET_INFO_BYTE = 0x50  [constexpr]
```

6.11.2.5 HEALTH_LEN

```
constexpr int rplidar::HEALTH_LEN = 3  [constexpr]
```

6.11.2.6 HEALTH_TYPE

```
constexpr int rplidar::HEALTH_TYPE = 6  [constexpr]
```

6.11.2.7 INFO_LEN

```
constexpr int rplidar::INFO_LEN = 20  [constexpr]
```

6.11.2.8 INFO_TYPE

```
constexpr int rplidar::INFO_TYPE = 4  [constexpr]
```

6.11.2.9 MAX_MOTOR_PWM

```
constexpr int rplidar::MAX_MOTOR_PWM = 1023  [constexpr]
```

6.11.2.10 RESET_BYTE

```
constexpr uint8_t rplidar::RESET_BYTE = 0x40  [constexpr]
```

6.11.2.11 SET_PWM_BYTE

```
constexpr uint8_t rplidar::SET_PWM_BYTE = 0xF0  [constexpr]
```

6.11.2.12 STOP_BYTE

```
constexpr uint8_t rplidar::STOP_BYTE = 0x25  [constexpr]
```

6.11.2.13 SYNC_BYTE

```
constexpr uint8_t rplidar::SYNC_BYTE = 0xA5  [constexpr]
```

6.11.2.14 SYNC_BYTE2

```
constexpr uint8_t rplidar::SYNC_BYTE2 = 0x5A  [constexpr]
```

6.12 utils Namespace Reference

Classes

- struct RawTypeNameFormat

Functions

- template<typename T >
  constexpr const auto & RawTypeName ()
- constexpr bool GetRawTypeNameFormat (RawTypeNameFormat ∗format)
- template<typename T >
  constexpr auto CexprTypeName ()
- template<typename T >
  const char ∗ TypeName ()
- template<typename T >
  const char ∗ TypeName (const T &)

6.12.1 Function Documentation

6.12.1.1 CexprTypeName()

```
template<typename T >
constexpr auto utils::CexprTypeName ( )  [constexpr]
```

6.12.1.2 GetRawTypeNameFormat()

```
constexpr bool utils::GetRawTypeNameFormat (
             RawTypeNameFormat ∗ format )  [inline], [constexpr]
```

6.12.1.3 RawTypeName()

```
template<typename T >
constexpr const auto & utils::RawTypeName ( )  [constexpr]
```

6.12.1.4 TypeName() [1/2]

```
template<typename T >
const char * utils::TypeName ( )
```

6.12.1.5 TypeName() [2/2]

```
template<typename T >
const char * utils::TypeName (
             const T &  )
```

Chapter 7

Class Documentation

7.1 behaviour_tree::BehaviourTree Class Reference

```
#include <BehaviourTree.hpp>
```

Public Member Functions

- BehaviourTree (unsigned int cycle_limit, std::vector< std::shared_ptr< Root > > roots)
- void start ()
- Status tick (const int tick_count, std::shared_ptr< Context > context)
- Status UseRoot (const int tick_count, std::shared_ptr< Context > context, const std::string &id)
- bool canRun () const
- void incrementCycle ()
- void resetCycles ()
- const std::string toString () const

Private Attributes

- int cycles = 0
- std::shared_ptr< Root > root_to_use
- const std::vector< std::shared_ptr< Root > > roots
- const unsigned int cycle_limit

7.1.1 Constructor & Destructor Documentation

7.1.1.1 BehaviourTree()

```
behaviour_tree::BehaviourTree::BehaviourTree (
            unsigned int cycle_limit,
            std::vector< std::shared_ptr< Root > > roots )  [inline]
```

7.1.2 Member Function Documentation

7.1.2.1 canRun()

```
bool behaviour_tree::BehaviourTree::canRun ( ) const  [inline]
```

7.1.2.2 incrementCycle()

```
void behaviour_tree::BehaviourTree::incrementCycle ( )  [inline]
```

7.1.2.3 resetCycles()

```
void behaviour_tree::BehaviourTree::resetCycles ( )  [inline]
```

7.1.2.4 start()

```
void behaviour_tree::BehaviourTree::start ( )  [inline]
```

7.1.2.5 tick()

```
Status behaviour_tree::BehaviourTree::tick (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline]
```

7.1.2.6 toString()

```
const std::string behaviour_tree::BehaviourTree::toString ( ) const  [inline]
```

7.1.2.7 UseRoot()

```
Status behaviour_tree::BehaviourTree::UseRoot (
            const int tick_count,
            std::shared_ptr< Context > context,
            const std::string & id )  [inline]
```

7.1.3 Member Data Documentation

7.1.3.1 cycle_limit

```
const unsigned int behaviour_tree::BehaviourTree::cycle_limit  [private]
```

7.1.3.2 cycles

```
int behaviour_tree::BehaviourTree::cycles = 0  [private]
```

7.1.3.3 root_to_use

```
std::shared_ptr<Root> behaviour_tree::BehaviourTree::root_to_use  [private]
```

7.1.3.4 roots

```
const std::vector<std::shared_ptr<Root> > behaviour_tree::BehaviourTree::roots  [private]
```

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

- behaviour_tree/include/behaviour_tree/BehaviourTree.hpp

7.2 behaviour_tree::BehaviourTreeParser Class Reference

```
#include <BehaviourTreeParser.hpp>
```

Public Member Functions

- BehaviourTreeParser (BehaviourTreeParser const &)=delete
- void operator= (BehaviourTreeParser const &)=delete
- void setCustomNodeParser (std::shared_ptr< CustomNodeParser > custom_node_parser)
- tl::expected< std::shared_ptr< BehaviourTree >, std::string > parseXML (const std::string &xml)
- tl::expected< std::shared_ptr< BehaviourTree >, std::string > parseFileXML (const std::string &file_path)

Static Public Member Functions

- static BehaviourTreeParser & instance ()

Private Member Functions

- BehaviourTreeParser ()
- tl::expected< std::shared_ptr< BehaviourTree >, std::string > parse (pugi::xml_document &doc)
- tl::expected< std::shared_ptr< Root >, std::string > parseRoot (const pugi::xml_node &node, const int index)
- tl::expected< std::shared_ptr< node::Node >, std::string > parseChild (const pugi::xml_node &node, const int index)
- tl::expected< std::shared_ptr< node::decorator::Decorator >, std::string > parseDecorator (const pugi↩ ::xml_node &node, const int index, const DecoratorType decorator_type)
- tl::expected< std::shared_ptr< node::blackboard::BlackboardNode >, std::string > parseBlackboardNode (const pugi::xml_node &node, const int index, const BlackboardType blackboard_node_type)
- tl::expected< std::shared_ptr< Composite >, std::string > parseComposite (const pugi::xml_node &node, const int index, const CompositeType composite_type)
- tl::expected< std::shared_ptr< node::custom::CustomNode >, std::string > parseCustomNode (const pugi::xml_node &node, const int index)

Private Attributes

- std::shared_ptr< CustomNodeParser > custom_node_parser

7.2.1 Constructor & Destructor Documentation

7.2.1.1 BehaviourTreeParser() [1/2]

```
behaviour_tree::BehaviourTreeParser::BehaviourTreeParser (
            BehaviourTreeParser const & )  [delete]
```

7.2.1.2 BehaviourTreeParser() [2/2]

```
behaviour_tree::BehaviourTreeParser::BehaviourTreeParser ( )  [inline], [private]
```

7.2.2 Member Function Documentation

7.2.2.1 instance()

```
static BehaviourTreeParser & behaviour_tree::BehaviourTreeParser::instance ( )  [inline],
[static]
```

7.2.2.2 operator=()

```
void behaviour_tree::BehaviourTreeParser::operator= (
            BehaviourTreeParser const &  )  [delete]
```

7.2.2.3 parse()

```
tl::expected< std::shared_ptr< BehaviourTree >, std::string > behaviour_tree::BehaviourTree←↩
Parser::parse (
            pugi::xml_document & doc )  [inline], [private]
```

7.2.2.4 parseBlackboardNode()

```
tl::expected< std::shared_ptr< node::blackboard::BlackboardNode >, std::string > behaviour_←↩
tree::BehaviourTreeParser::parseBlackboardNode (
            const pugi::xml_node & node,
            const int index,
            const BlackboardType blackboard_node_type )  [inline], [private]
```

7.2.2.5 parseChild()

```
tl::expected< std::shared_ptr< node::Node >, std::string > behaviour_tree::BehaviourTree←↩
Parser::parseChild (
            const pugi::xml_node & node,
            const int index )  [inline], [private]
```

7.2.2.6 parseComposite()

```
tl::expected< std::shared_ptr< Composite >, std::string > behaviour_tree::BehaviourTree←↩
Parser::parseComposite (
            const pugi::xml_node & node,
            const int index,
            const CompositeType composite_type )  [inline], [private]
```

7.2.2.7 parseCustomNode()

```
tl::expected< std::shared_ptr< node::custom::CustomNode >, std::string > behaviour_tree::←↩
BehaviourTreeParser::parseCustomNode (
            const pugi::xml_node & node,
            const int index )  [inline], [private]
```

7.2.2.8 parseDecorator()

```
tl::expected< std::shared_ptr< node::decorator::Decorator >, std::string > behaviour_tree::↩
BehaviourTreeParser::parseDecorator (
            const pugi::xml_node & node,
            const int index,
            const DecoratorType decorator_type ) [inline], [private]
```

7.2.2.9 parseFileXML()

```
tl::expected< std::shared_ptr< BehaviourTree >, std::string > behaviour_tree::BehaviourTree↩
Parser::parseFileXML (
            const std::string & file_path ) [inline]
```

7.2.2.10 parseRoot()

```
tl::expected< std::shared_ptr< Root >, std::string > behaviour_tree::BehaviourTreeParser↩
::parseRoot (
            const pugi::xml_node & node,
            const int index ) [inline], [private]
```

7.2.2.11 parseXML()

```
tl::expected< std::shared_ptr< BehaviourTree >, std::string > behaviour_tree::BehaviourTree↩
Parser::parseXML (
            const std::string & xml ) [inline]
```

7.2.2.12 setCustomNodeParser()

```
void behaviour_tree::BehaviourTreeParser::setCustomNodeParser (
            std::shared_ptr< CustomNodeParser > custom_node_parser ) [inline]
```

7.2.3 Member Data Documentation

7.2.3.1 custom_node_parser

```
std::shared_ptr<CustomNodeParser> behaviour_tree::BehaviourTreeParser::custom_node_parser
[private]
```

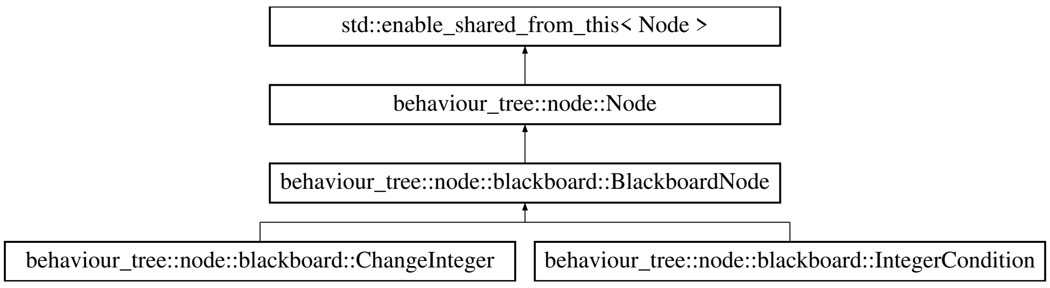
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/BehaviourTreeParser.hpp

7.3 behaviour_tree::node::blackboard::BlackboardNode Class Reference

```
#include <BlackboardNode.hpp>
```

Inheritance diagram for behaviour_tree::node::blackboard::BlackboardNode:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::blackboard::BlackboardNode
behaviour_tree::node::blackboard::ChangeInteger
behaviour_tree::node::blackboard::IntegerCondition

Public Member Functions

- BlackboardNode (const std::string &name)
- virtual const BlackboardType type () const =0

Additional Inherited Members

7.3.1 Constructor & Destructor Documentation

7.3.1.1 BlackboardNode()

```
behaviour_tree::node::blackboard::BlackboardNode::BlackboardNode (
             const std::string & name )  [inline]
```

7.3.2 Member Function Documentation

7.3.2.1 type()

```
virtual const BlackboardType behaviour_tree::node::blackboard::BlackboardNode::type ( ) const
[pure virtual]
```

Implemented in behaviour_tree::node::blackboard::ChangeInteger, and behaviour_tree::node::blackboard::IntegerCondition.

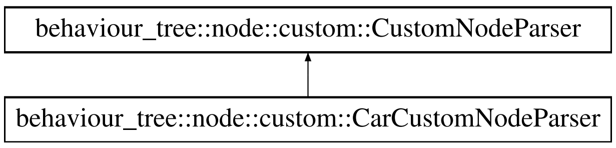
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/blackboard/BlackboardNode.hpp

7.4 behaviour_tree::node::custom::CarCustomNodeParser Class Reference

```
#include <CarCustomNodeParser.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::CarCustomNodeParser:


behaviour_tree::node::custom::CustomNodeParser
behaviour_tree::node::custom::CarCustomNodeParser

Public Member Functions

- CarCustomNodeParser ()
- tl::expected< std::shared_ptr< custom::CustomNode >, std::string > parseCustomNode (const pugi::xml↩ _node &node, const int index) override

7.4.1 Constructor & Destructor Documentation

7.4.1.1 CarCustomNodeParser()

```
behaviour_tree::node::custom::CarCustomNodeParser::CarCustomNodeParser ( )  [inline]
```

7.4.2 Member Function Documentation

7.4.2.1 parseCustomNode()

```
tl::expected< std::shared_ptr< custom::CustomNode >, std::string > behaviour_tree::node↵
::custom::CarCustomNodeParser::parseCustomNode (
            const pugi::xml_node & node,
            const int index )  [inline], [override], [virtual]
```

Implements behaviour_tree::node::custom::CustomNodeParser.

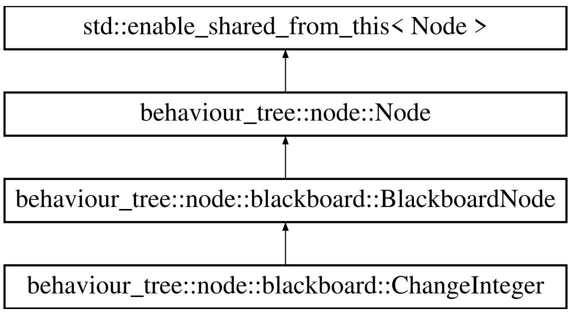
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/CarCustomNodeParser.hpp

7.5 behaviour_tree::node::blackboard::ChangeInteger Class Reference

```
#include <ChangeInteger.hpp>
```

Inheritance diagram for behaviour_tree::node::blackboard::ChangeInteger:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::blackboard::BlackboardNode
behaviour_tree::node::blackboard::ChangeInteger

Public Member Functions

- ChangeInteger (const std::string &name, const std::string variable_name, const IntegerChangeType integer_change_type, const int value)
- const BlackboardType type () const final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override

Private Attributes

- const std::string variable_name
- const IntegerChangeType integer_change_type
- const int value

Additional Inherited Members

7.5.1 Constructor & Destructor Documentation

7.5.1.1 ChangeInteger()

```
behaviour_tree::node::blackboard::ChangeInteger::ChangeInteger (
            const std::string & name,
            const std::string variable_name,
            const IntegerChangeType integer_change_type,
            const int value ) [inline]
```

7.5.2 Member Function Documentation

7.5.2.1 run()

```
const Status behaviour_tree::node::blackboard::ChangeInteger::run (
            const int tick_count,
            std::shared_ptr< Context > context ) [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.5.2.2 toString()

```
const std::string behaviour_tree::node::blackboard::ChangeInteger::toString ( ) const [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.5.2.3 type()

```
const BlackboardType behaviour_tree::node::blackboard::ChangeInteger::type ( ) const [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::blackboard::BlackboardNode.

7.5.3 Member Data Documentation

7.5.3.1 integer_change_type

```
const IntegerChangeType behaviour_tree::node::blackboard::ChangeInteger::integer_change_type
[private]
```

7.5.3.2 value

```
const int behaviour_tree::node::blackboard::ChangeInteger::value  [private]
```

7.5.3.3 variable_name

```
const std::string behaviour_tree::node::blackboard::ChangeInteger::variable_name  [private]
```

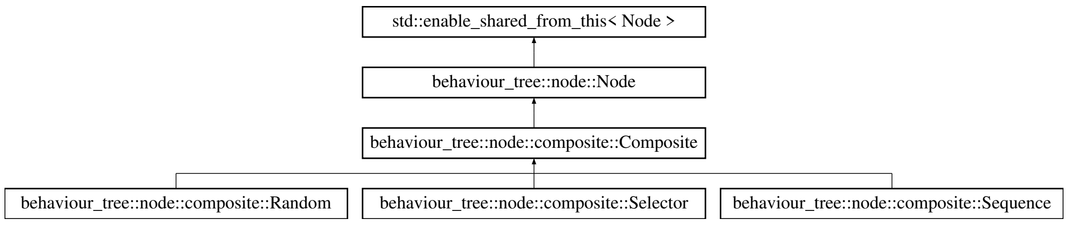
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/blackboard/ChangeInteger.hpp

7.6 behaviour_tree::node::composite::Composite Class Reference

```
#include <Composite.hpp>
```

Inheritance diagram for behaviour_tree::node::composite::Composite:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::composite::Composite
behaviour_tree::node::composite::Random
behaviour_tree::node::composite::Selector
behaviour_tree::node::composite::Sequence

Public Member Functions

- Composite (const std::string &name, std::vector< std::shared_ptr< Node > > children)
- Status tick (const int tick_count, std::shared_ptr< Context > context, const int start_index)
- virtual const Status run (const int tick_count, std::shared_ptr< Context > context, const int start_index)=0
- virtual const CompositeType type () const =0

Protected Attributes

- std::vector< std::shared_ptr< Node > > children

7.6.1 Constructor & Destructor Documentation

7.6.1.1 Composite()

```
behaviour_tree::node::composite::Composite::Composite (
            const std::string & name,
            std::vector< std::shared_ptr< Node > > children )  [inline]
```

7.6.2 Member Function Documentation

7.6.2.1 run()

```
virtual const Status behaviour_tree::node::composite::Composite::run (
            const int tick_count,
            std::shared_ptr< Context > context,
            const int start_index )  [pure virtual]
```

Implemented in behaviour_tree::node::composite::Random, behaviour_tree::node::composite::Selector, and behaviour_tree::node::composite::Sequence.

7.6.2.2 tick()

```
Status behaviour_tree::node::composite::Composite::tick (
            const int tick_count,
            std::shared_ptr< Context > context,
            const int start_index )  [inline]
```

7.6.2.3 type()

```
virtual const CompositeType behaviour_tree::node::composite::Composite::type ( ) const  [pure
virtual]
```

Implemented in behaviour_tree::node::composite::Random, behaviour_tree::node::composite::Selector, and behaviour_tree::node::composite::Sequence.

7.6.3 Member Data Documentation

7.6.3.1 children

```
std::vector<std::shared_ptr<Node> > behaviour_tree::node::composite::Composite::children
[protected]
```

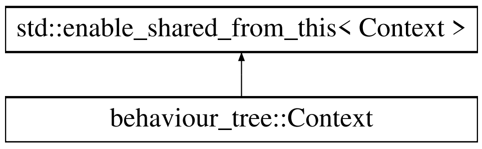
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/composite/Composite.hpp

7.7 behaviour_tree::Context Class Reference

```
#include <Context.h>
```

Inheritance diagram for behaviour_tree::Context:



Public Member Functions

- Context (std::shared_ptr< BehaviourTree > behaviour_tree)
- void update (const int tick_count)
- Status UseRoot (const int tick_count, const std::string &id)
- std::shared_ptr< BehaviourTree > getBehaviourTree () const
- void pushNodeTrace (std::pair< std::shared_ptr< node::Node >, int > node_trace)
- void popNode ()
- bool canRun () const
- void setBlackboard (const std::string &key, const int value)
- int getBlackboard (const std::string &key) const
- int containsBlackboard (const std::string &key) const
- virtual void _ ()

Private Attributes

- std::shared_ptr< BehaviourTree > behaviour_tree
- std::vector< std::pair< std::shared_ptr< node::Node >, int > > node_trace_list
- std::unordered_map< std::string, int > blackboard

7.7.1 Constructor & Destructor Documentation

7.7.1.1 Context()

```
behaviour_tree::Context::Context (
             std::shared_ptr< BehaviourTree > behaviour_tree )
```

7.7.2 Member Function Documentation

7.7.2.1 _()

```
virtual void behaviour_tree::Context::_ ( )  [inline], [virtual]
```

7.7.2.2 canRun()

```
bool behaviour_tree::Context::canRun ( ) const
```

7.7.2.3 containsBlackboard()

```
int behaviour_tree::Context::containsBlackboard (
             const std::string & key ) const
```

7.7.2.4 getBehaviourTree()

```
std::shared_ptr< BehaviourTree > behaviour_tree::Context::getBehaviourTree ( ) const  [inline]
```

7.7.2.5 getBlackboard()

```
int behaviour_tree::Context::getBlackboard (
             const std::string & key ) const
```

7.7.2.6 popNode()

```
void behaviour_tree::Context::popNode ( )
```

7.7.2.7 pushNodeTrace()

```
void behaviour_tree::Context::pushNodeTrace (
            std::pair< std::shared_ptr< node::Node >, int > node_trace )
```

7.7.2.8 setBlackboard()

```
void behaviour_tree::Context::setBlackboard (
            const std::string & key,
            const int value )
```

7.7.2.9 update()

```
void behaviour_tree::Context::update (
            const int tick_count )
```

7.7.2.10 UseRoot()

```
Status behaviour_tree::Context::UseRoot (
            const int tick_count,
            const std::string & id )
```

7.7.3 Member Data Documentation

7.7.3.1 behaviour_tree

```
std::shared_ptr<BehaviourTree> behaviour_tree::Context::behaviour_tree  [private]
```

7.7.3.2 blackboard

```
std::unordered_map<std::string, int> behaviour_tree::Context::blackboard  [private]
```

7.7.3.3 node_trace_list

```
std::vector<std::pair<std::shared_ptr<node::Node>, int> > behaviour_tree::Context::node_↩
trace_list  [private]
```

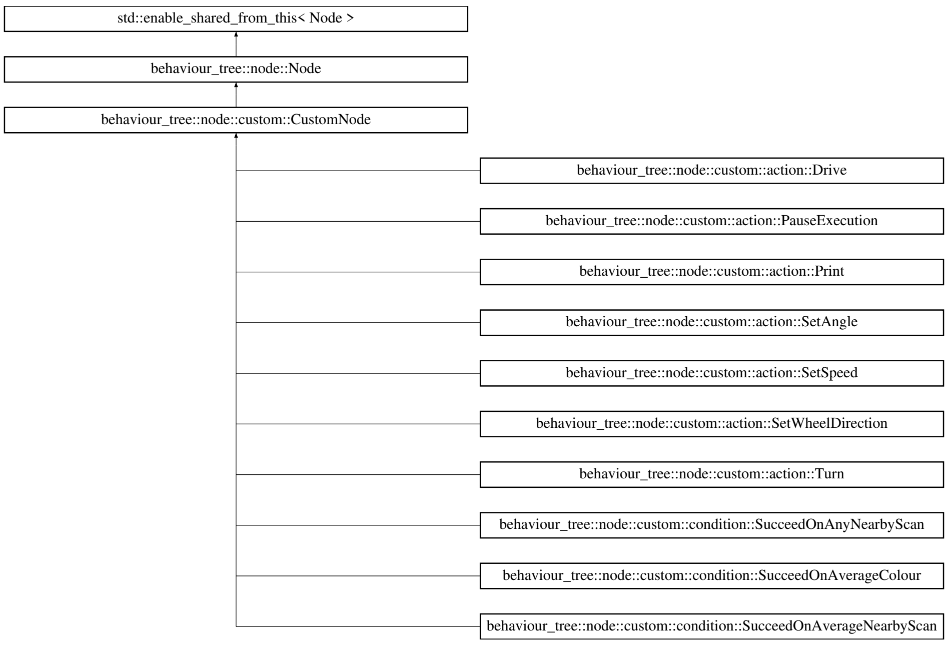
The documentation for this class was generated from the following files:

- behaviour_tree/include/behaviour_tree/Context.h
- behaviour_tree/include/behaviour_tree/Context.cpp

7.8 behaviour_tree::node::custom::CustomNode Class Reference

```
#include <CustomNode.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::CustomNode:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::Drive
behaviour_tree::node::custom::action::PauseExecution
behaviour_tree::node::custom::action::Print
behaviour_tree::node::custom::action::SetAngle
behaviour_tree::node::custom::action::SetSpeed
behaviour_tree::node::custom::action::SetWheelDirection
behaviour_tree::node::custom::action::Turn
behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan
behaviour_tree::node::custom::condition::SucceedOnAverageColour
behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan

Public Member Functions

- CustomNode (const std::string &name)

Additional Inherited Members

7.8.1 Constructor & Destructor Documentation

7.8.1.1 CustomNode()

```
behaviour_tree::node::custom::CustomNode::CustomNode (
             const std::string & name )  [inline]
```

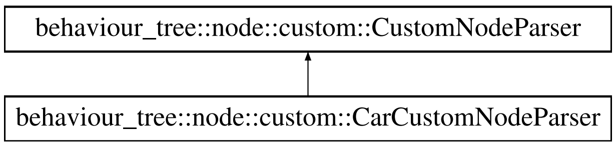
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/custom/CustomNode.hpp

7.9 behaviour_tree::node::custom::CustomNodeParser Class Reference

```
#include <CustomNodeParser.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::CustomNodeParser:


behaviour_tree::node::custom::CustomNodeParser
behaviour_tree::node::custom::CarCustomNodeParser

Public Member Functions

- virtual tl::expected< std::shared_ptr< CustomNode >, std::string > parseCustomNode (const pugi::xml_node &node, const int index)=0

7.9.1 Member Function Documentation

7.9.1.1 parseCustomNode()

```
virtual tl::expected< std::shared_ptr< CustomNode >, std::string > behaviour_tree::node
::custom::CustomNodeParser::parseCustomNode (
             const pugi::xml_node & node,
             const int index )  [pure virtual]
```

Implemented in behaviour_tree::node::custom::CarCustomNodeParser.

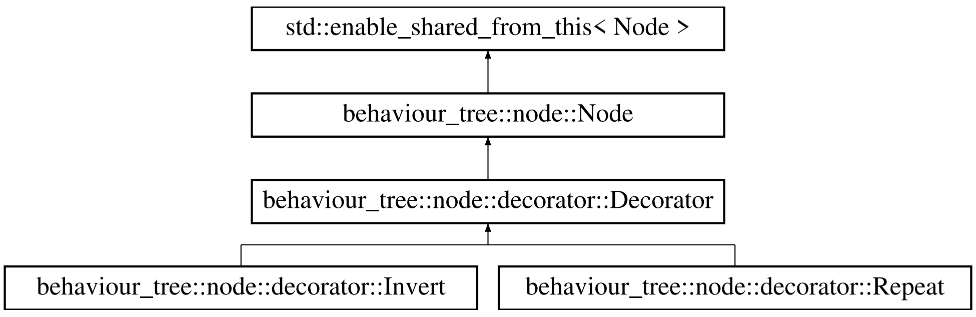
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/custom/CustomNodeParser.hpp

7.10 behaviour_tree::node::decorator::Decorator Class Reference

```
#include <Decorator.hpp>
```

Inheritance diagram for behaviour_tree::node::decorator::Decorator:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::decorator::Decorator
behaviour_tree::node::decorator::Invert
behaviour_tree::node::decorator::Repeat

Public Member Functions

- Decorator (const std::string &name, std::shared_ptr< Node > child)
- virtual const DecoratorType type () const =0

Protected Attributes

- std::shared_ptr< Node > child

7.10.1 Constructor & Destructor Documentation

7.10.1.1 Decorator()

```
behaviour_tree::node::decorator::Decorator::Decorator (
             const std::string & name,
             std::shared_ptr< Node > child )  [inline]
```

7.10.2 Member Function Documentation

7.10.2.1 type()

```
virtual const DecoratorType behaviour_tree::node::decorator::Decorator::type ( ) const  [pure
virtual]
```

Implemented in behaviour_tree::node::decorator::Invert, and behaviour_tree::node::decorator::Repeat.

7.10.3 Member Data Documentation

7.10.3.1 child

```
std::shared_ptr<Node> behaviour_tree::node::decorator::Decorator::child  [protected]
```

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

- behaviour_tree/include/behaviour_tree/node/decorator/Decorator.hpp

7.11 rplidar::DeviceInfo Struct Reference

```
#include <RPLidar.h>
```

Public Attributes

- uint8_t model
- std::pair< uint8_t, uint8_t > firmware
- uint8_t hardware
- std::string serialNumber

7.11.1 Member Data Documentation

7.11.1.1 firmware

```
std::pair<uint8_t, uint8_t> rplidar::DeviceInfo::firmware
```

7.11.1.2 hardware

```
uint8_t rplidar::DeviceInfo::hardware
```

7.11.1.3 model

```
uint8_t rplidar::DeviceInfo::model
```

7.11.1.4 serialNumber

```
std::string rplidar::DeviceInfo::serialNumber
```

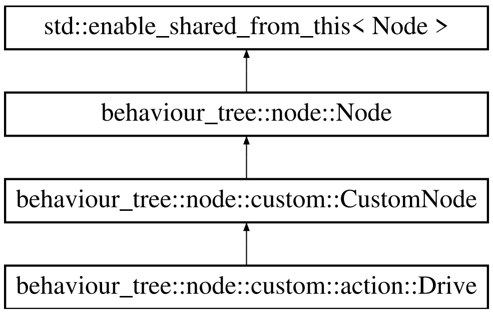
The documentation for this struct was generated from the following file:

- rplidar/include/RPLidar.h

7.12 behaviour_tree::node::custom::action::Drive Class Reference

```
#include <Drive.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::action::Drive:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::Drive

Public Member Functions

- Drive (const std::string &name, const int speed, const DirectionType direction_type)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override
- const int getSpeed () const
- const DirectionType getDirectionType () const

Static Public Member Functions

- static const tl::expected< std::shared_ptr< Drive >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const int speed
- const DirectionType direction_type

Additional Inherited Members

7.12.1 Constructor & Destructor Documentation

7.12.1.1 Drive()

```
behaviour_tree::node::custom::action::Drive::Drive (
             const std::string & name,
             const int speed,
             const DirectionType direction_type )  [inline]
```

7.12.2 Member Function Documentation

7.12.2.1 getDirectionType()

```
const DirectionType behaviour_tree::node::custom::action::Drive::getDirectionType ( ) const
[inline]
```

7.12.2.2 getSpeed()

```
const int behaviour_tree::node::custom::action::Drive::getSpeed ( ) const  [inline]
```

7.12.2.3 parse()

```
static const tl::expected< std::shared_ptr< Drive >, std::string > behaviour_tree::node←
::custom::action::Drive::parse (
             const pugi::xml_node & node,
             const int index,
             const std::string & name_attribute )  [inline], [static]
```

7.12.2.4 run()

```
const Status behaviour_tree::node::custom::action::Drive::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.12.2.5 toString()

```
const std::string behaviour_tree::node::custom::action::Drive::toString ( ) const  [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.12.3 Member Data Documentation

7.12.3.1 direction_type

```
const DirectionType behaviour_tree::node::custom::action::Drive::direction_type  [private]
```

7.12.3.2 speed

```
const int behaviour_tree::node::custom::action::Drive::speed  [private]
```

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

- behaviour_tree_car/include/behaviour_tree/node/custom/action/Drive.hpp

7.13 rplidar::ExpressPacket Class Reference

```
#include <RPLidar.h>
```

Public Member Functions

- ExpressPacket (std::vector< uint8_t > data)

Static Public Member Functions

- static int getSign (uint8_t value)

Public Attributes

- std::vector< uint16_t > distance
- std::vector< float > angle
- bool new_scan
- float start_angle

Static Public Attributes

- static const uint8_t sync1 = 0xa
- static const uint8_t sync2 = 0x5

7.13.1 Constructor & Destructor Documentation

7.13.1.1 ExpressPacket()

```
rplidar::ExpressPacket::ExpressPacket (
            std::vector< uint8_t > data )  [inline]
```

7.13.2 Member Function Documentation

7.13.2.1 getSign()

```
static int rplidar::ExpressPacket::getSign (
            uint8_t value )  [inline], [static]
```

7.13.3 Member Data Documentation

7.13.3.1 angle

```
std::vector<float> rplidar::ExpressPacket::angle
```

7.13.3.2 distance

```
std::vector<uint16_t> rplidar::ExpressPacket::distance
```

7.13.3.3 new_scan

```
bool rplidar::ExpressPacket::new_scan
```

7.13.3.4 start_angle

```
float rplidar::ExpressPacket::start_angle
```

7.13.3.5 sync1

```
const uint8_t rplidar::ExpressPacket::sync1 = 0xa  [static]
```

7.13.3.6 sync2

```
const uint8_t rplidar::ExpressPacket::sync2 = 0x5  [static]
```

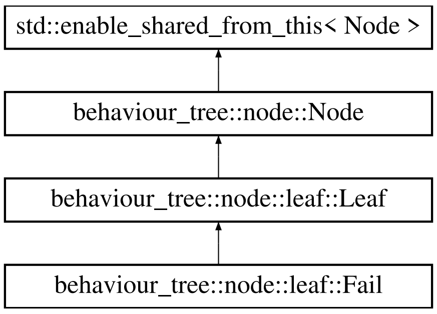
The documentation for this class was generated from the following file:

- rplidar/include/RPLidar.h

7.14 behaviour_tree::node::leaf::Fail Class Reference

```
#include <Fail.hpp>
```

Inheritance diagram for behaviour_tree::node::leaf::Fail:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::leaf::Leaf
behaviour_tree::node::leaf::Fail

Public Member Functions

- Fail (const std::string &name)
- const LeafType type () const final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override

Additional Inherited Members

7.14.1 Constructor & Destructor Documentation

7.14.1.1 Fail()

```
behaviour_tree::node::leaf::Fail::Fail (
            const std::string & name )  [inline]
```

7.14.2 Member Function Documentation

7.14.2.1 run()

```
const Status behaviour_tree::node::leaf::Fail::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.14.2.2 toString()

```
const std::string behaviour_tree::node::leaf::Fail::toString ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::Node.

7.14.2.3 type()

```
const LeafType behaviour_tree::node::leaf::Fail::type ( ) const  [inline], [final], [override],
[virtual]
```

Implements behaviour_tree::node::leaf::Leaf.

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

- behaviour_tree/include/behaviour_tree/node/leaf/Fail.hpp

7.15 rplidar::HealthInfo Struct Reference

Health Info for Lidar Scanner.

```
#include <RPLidar.h>
```

Public Attributes

- std::string status

  'Good', 'Warning' or 'Error' statuses
- int errorCode

  The related error code that caused a warning/error.

7.15.1 Detailed Description

Health Info for Lidar Scanner.

7.15.2 Member Data Documentation

7.15.2.1 errorCode

```
int rplidar::HealthInfo::errorCode
```

The related error code that caused a warning/error.

7.15.2.2 status

```
std::string rplidar::HealthInfo::status
```

'Good', 'Warning' or 'Error' statuses

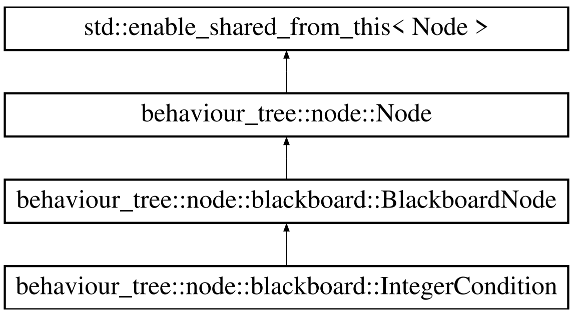
The documentation for this struct was generated from the following file:

- rplidar/include/RPLidar.h

7.16 behaviour_tree::node::blackboard::IntegerCondition Class Reference

```
#include <IntegerCondition.hpp>
```

Inheritance diagram for behaviour_tree::node::blackboard::IntegerCondition:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::blackboard::BlackboardNode
behaviour_tree::node::blackboard::IntegerCondition

Public Member Functions

- IntegerCondition (const std::string &name, const std::string variable_name, const ConditionOperatorType condition_operator, const int value)
- const BlackboardType type () const final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override

Private Attributes

- const std::string variable_name
- const ConditionOperatorType condition_operator
- const int value

Additional Inherited Members

7.16.1 Constructor & Destructor Documentation

7.16.1.1 IntegerCondition()

```
behaviour_tree::node::blackboard::IntegerCondition::IntegerCondition (
             const std::string & name,
             const std::string variable_name,
             const ConditionOperatorType condition_operator,
             const int value ) [inline]
```

7.16.2 Member Function Documentation

7.16.2.1 run()

```
const Status behaviour_tree::node::blackboard::IntegerCondition::run (
             const int tick_count,
             std::shared_ptr< Context > context ) [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.16.2.2 toString()

```
const std::string behaviour_tree::node::blackboard::IntegerCondition::toString ( ) const [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.16.2.3 type()

```
const BlackboardType behaviour_tree::node::blackboard::IntegerCondition::type ( ) const  [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::blackboard::BlackboardNode.

7.16.3 Member Data Documentation

7.16.3.1 condition_operator

```
const ConditionOperatorType behaviour_tree::node::blackboard::IntegerCondition::condition_↩
operator  [private]
```

7.16.3.2 value

```
const int behaviour_tree::node::blackboard::IntegerCondition::value  [private]
```

7.16.3.3 variable_name

```
const std::string behaviour_tree::node::blackboard::IntegerCondition::variable_name  [private]
```

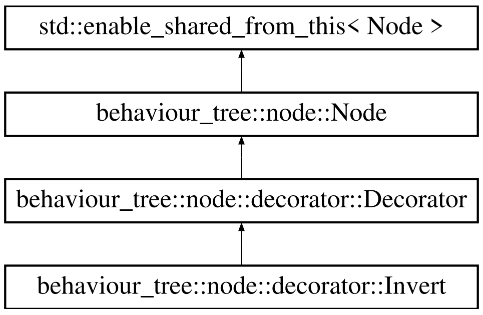
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/blackboard/IntegerCondition.hpp

7.17 behaviour_tree::node::decorator::Invert Class Reference

```
#include <Invert.hpp>
```

Inheritance diagram for behaviour_tree::node::decorator::Invert:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::decorator::Decorator
behaviour_tree::node::decorator::Invert

Public Member Functions

- Invert (const std::string &name, std::shared_ptr< Node > child)
- const DecoratorType type () const final override
- void start (std::shared_ptr< Context > context) final override
- void finish (std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override

Additional Inherited Members

7.17.1 Constructor & Destructor Documentation

7.17.1.1 Invert()

```
behaviour_tree::node::decorator::Invert::Invert (
            const std::string & name,
            std::shared_ptr< Node > child )  [inline]
```

7.17.2 Member Function Documentation

7.17.2.1 finish()

```
void behaviour_tree::node::decorator::Invert::finish (
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Reimplemented from behaviour_tree::node::Node.

7.17.2.2 run()

```
const Status behaviour_tree::node::decorator::Invert::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.17.2.3 start()

```
void behaviour_tree::node::decorator::Invert::start (
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Reimplemented from behaviour_tree::node::Node.

7.17.2.4 toString()

```
const std::string behaviour_tree::node::decorator::Invert::toString ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::Node.

7.17.2.5 type()

```
const DecoratorType behaviour_tree::node::decorator::Invert::type ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::decorator::Decorator.

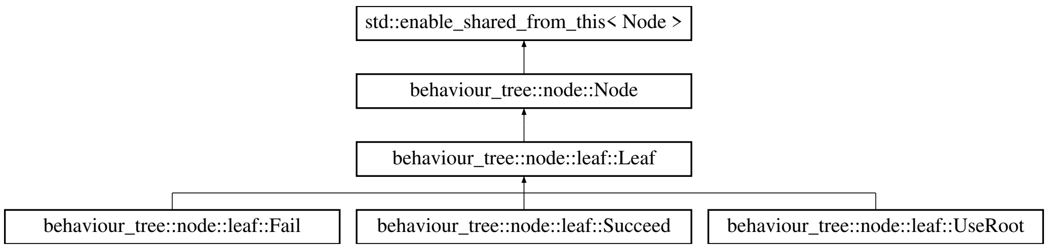
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/decorator/Invert.hpp

7.18 behaviour_tree::node::leaf::Leaf Class Reference

```
#include <Leaf.hpp>
```

Inheritance diagram for behaviour_tree::node::leaf::Leaf:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::leaf::Leaf
behaviour_tree::node::leaf::Fail
behaviour_tree::node::leaf::Succeed
behaviour_tree::node::leaf::UseRoot

Public Member Functions

- Leaf (const std::string &name)
- virtual const LeafType type () const =0

Additional Inherited Members

7.18.1 Constructor & Destructor Documentation

7.18.1.1 Leaf()

```
behaviour_tree::node::leaf::Leaf::Leaf (
             const std::string & name )  [inline]
```

7.18.2 Member Function Documentation

7.18.2.1 type()

```
virtual const LeafType behaviour_tree::node::leaf::Leaf::type ( ) const  [pure virtual]
```

Implemented in behaviour_tree::node::leaf::Fail, behaviour_tree::node::leaf::Succeed, and behaviour_tree::node::leaf::UseRoot.

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

- behaviour_tree/include/behaviour_tree/node/leaf/Leaf.hpp

7.19 rplidar::Measure Struct Reference

```
#include <RPLidar.h>
```

Public Attributes

- bool newScan
- int quality
- double angle
- double distance

7.19.1 Member Data Documentation

7.19.1.1 angle

```
double rplidar::Measure::angle
```

7.19.1.2 distance

```
double rplidar::Measure::distance
```

7.19.1.3 newScan

```
bool rplidar::Measure::newScan
```

7.19.1.4 quality

```
int rplidar::Measure::quality
```

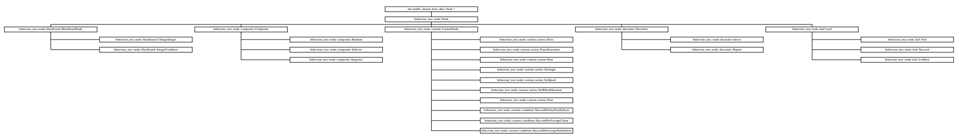
The documentation for this struct was generated from the following file:

- rplidar/include/RPLidar.h

7.20 behaviour_tree::node::Node Class Reference

```
#include <Node.hpp>
```

Inheritance diagram for behaviour_tree::node::Node:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::blackboard::BlackboardNode
behaviour_tree::node::blackboard::ChangeInteger
behaviour_tree::node::blackboard::IntegerCondition
behaviour_tree::node::composite::Composite
behaviour_tree::node::composite::Random
behaviour_tree::node::composite::Selector
behaviour_tree::node::composite::Sequence
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::Drive
behaviour_tree::node::custom::action::PauseExecution
behaviour_tree::node::custom::action::Print
behaviour_tree::node::custom::action::SetAngle
behaviour_tree::node::custom::action::SetSpeed
behaviour_tree::node::custom::action::SetWheelDirection
behaviour_tree::node::custom::action::Turn
behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan
behaviour_tree::node::custom::condition::SucceedOnAverageColour
behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan
behaviour_tree::node::decorator::Decorator
behaviour_tree::node::decorator::Invert
behaviour_tree::node::decorator::Repeat
behaviour_tree::node::leaf::Leaf
behaviour_tree::node::leaf::Fail
behaviour_tree::node::leaf::Succeed
behaviour_tree::node::leaf::UseRoot

Public Member Functions

- Node (const std::string &name)
- virtual void start (std::shared_ptr< Context > context)
- virtual const Status run (const int tick_count, std::shared_ptr< Context > context)=0
- virtual void finish (std::shared_ptr< Context > context)
- Status tick (const int tick_count, std::shared_ptr< Context > context)
- virtual const std::string toString () const =0
- const std::string & getName () const

Protected Attributes

- const std::string name
- bool started = false

7.20.1 Constructor & Destructor Documentation

7.20.1.1 Node()

```
behaviour_tree::node::Node::Node (
            const std::string & name )  [inline]
```

7.20.2 Member Function Documentation

7.20.2.1 finish()

```
virtual void behaviour_tree::node::Node::finish (
            std::shared_ptr< Context > context )  [inline], [virtual]
```

Reimplemented in behaviour_tree::node::decorator::Invert, and behaviour_tree::node::custom::action::PauseExecution.

7.20.2.2 getName()

```
const std::string & behaviour_tree::node::Node::getName ( ) const  [inline]
```

7.20.2.3 run()

```
virtual const Status behaviour_tree::node::Node::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [pure virtual]
```

Implemented in behaviour_tree::node::blackboard::ChangeInteger, behaviour_tree::node::blackboard::IntegerCondition, behaviour_tree::node::composite::Random, behaviour_tree::node::composite::Selector, behaviour_tree::node::composite::Sequence, behaviour_tree::node::decorator::Invert, behaviour_tree::node::decorator::Repeat, behaviour_tree::node::leaf::Fail, behaviour_tree::node::leaf::Succeed, behaviour_tree::node::leaf::UseRoot, behaviour_tree::node::custom::action::Drive, behaviour_tree::node::custom::action::PauseExecution, behaviour_tree::node::custom::action::Print, behaviour_tree::node::custom::a
behaviour_tree::node::custom::action::SetSpeed, behaviour_tree::node::custom::action::SetWheelDirection, behaviour_tree::node::custom::action::Turn, behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan, behaviour_tree::node::custom::condition::SucceedOnAverageColour, and behaviour_tree::node::custom::condition::SucceedOnAvera

7.20.2.4 start()

```
virtual void behaviour_tree::node::Node::start (
             std::shared_ptr< Context > context )  [inline], [virtual]
```

Reimplemented in behaviour_tree::node::composite::Selector, behaviour_tree::node::composite::Sequence, behaviour_tree::node::decorator::Invert, behaviour_tree::node::decorator::Repeat, and behaviour_tree::node::custom::action::PauseE

7.20.2.5 tick()

```
Status behaviour_tree::node::Node::tick (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline]
```

7.20.2.6 toString()

```
virtual const std::string behaviour_tree::node::Node::toString ( ) const  [pure virtual]
```

Implemented in behaviour_tree::node::blackboard::ChangeInteger, behaviour_tree::node::blackboard::IntegerCondition, behaviour_tree::node::composite::Random, behaviour_tree::node::composite::Selector, behaviour_tree::node::composite::Sequence, behaviour_tree::node::decorator::Invert, behaviour_tree::node::decorator::Repeat, behaviour_tree::node::leaf::Fail, behaviour_tree::node::leaf::Succeed, behaviour_tree::node::leaf::UseRoot, behaviour_tree::node::custom::action::Drive, behaviour_tree::node::custom::action::PauseExecution, behaviour_tree::node::custom::action::Print, behaviour_tree::node::custom::a behaviour_tree::node::custom::action::SetSpeed, behaviour_tree::node::custom::action::SetWheelDirection, behaviour_tree::node::custom::action::Turn, behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan, behaviour_tree::node::custom::condition::SucceedOnAverageColour, and behaviour_tree::node::custom::condition::SucceedOnAvera

7.20.3 Member Data Documentation

7.20.3.1 name

```
const std::string behaviour_tree::node::Node::name  [protected]
```

7.20.3.2 started

```
bool behaviour_tree::node::Node::started = false  [protected]
```

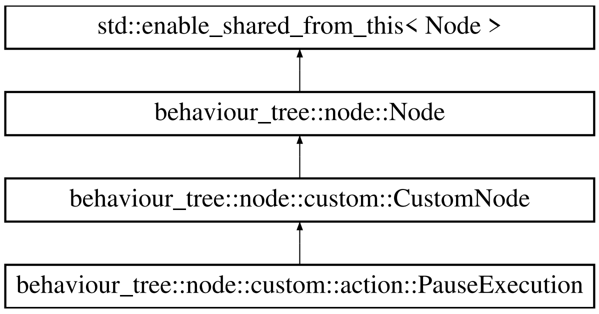
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/Node.hpp

7.21 behaviour_tree::node::custom::action::PauseExecution Class Reference

```
#include <PauseExecution.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::action::PauseExecution:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::PauseExecution

Public Member Functions

- PauseExecution (const std::string &name, const int ms)
- void start (std::shared_ptr< Context > context) final override
- void finish (std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override
- const int getMS () const

Static Public Member Functions

- static const tl::expected< std::shared_ptr< PauseExecution >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- std::chrono::steady_clock::time_point start_time
- const int ms

Additional Inherited Members

7.21.1 Constructor & Destructor Documentation

7.21.1.1 PauseExecution()

```
behaviour_tree::node::custom::action::PauseExecution::PauseExecution (
             const std::string & name,
             const int ms )  [inline]
```

7.21.2 Member Function Documentation

7.21.2.1 finish()

```
void behaviour_tree::node::custom::action::PauseExecution::finish (
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Reimplemented from behaviour_tree::node::Node.

7.21.2.2 getMS()

```
const int behaviour_tree::node::custom::action::PauseExecution::getMS ( ) const  [inline]
```

7.21.2.3 parse()

```
static const tl::expected< std::shared_ptr< PauseExecution >, std::string > behaviour_tree↩
::node::custom::action::PauseExecution::parse (
            const pugi::xml_node & node,
            const int index,
            const std::string & name_attribute )  [inline], [static]
```

7.21.2.4 run()

```
const Status behaviour_tree::node::custom::action::PauseExecution::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.21.2.5 start()

```
void behaviour_tree::node::custom::action::PauseExecution::start (
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Reimplemented from behaviour_tree::node::Node.

7.21.2.6 toString()

```
const std::string behaviour_tree::node::custom::action::PauseExecution::toString ( ) const
[inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.21.3 Member Data Documentation

7.21.3.1 ms

```
const int behaviour_tree::node::custom::action::PauseExecution::ms  [private]
```

7.21.3.2 start_time

```
std::chrono::steady_clock::time_point behaviour_tree::node::custom::action::PauseExecution↩
::start_time  [private]
```

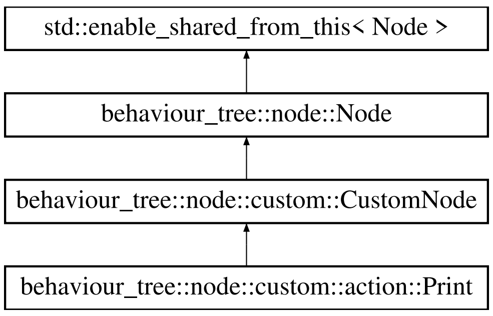
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/action/PauseExecution.hpp

7.22 behaviour_tree::node::custom::action::Print Class Reference

```
#include <Print.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::action::Print:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::Print

Public Member Functions

- Print (const std::string &name, const std::string &text)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string & getText () const
- const std::string toString () const final override

Static Public Member Functions

- static const tl::expected< std::shared_ptr< Print >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const std::string text

Additional Inherited Members

7.22.1 Constructor & Destructor Documentation

7.22.1.1 Print()

```
behaviour_tree::node::custom::action::Print::Print (
            const std::string & name,
            const std::string & text )  [inline]
```

7.22.2 Member Function Documentation

7.22.2.1 getText()

```
const std::string & behaviour_tree::node::custom::action::Print::getText ( ) const  [inline]
```

7.22.2.2 parse()

```
static const tl::expected< std::shared_ptr< Print >, std::string > behaviour_tree::node↩
::custom::action::Print::parse (
            const pugi::xml_node & node,
            const int index,
            const std::string & name_attribute )  [inline], [static]
```

7.22.2.3 run()

```
const Status behaviour_tree::node::custom::action::Print::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.22.2.4 toString()

```
const std::string behaviour_tree::node::custom::action::Print::toString ( ) const  [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.22.3 Member Data Documentation

7.22.3.1 text

```
const std::string behaviour_tree::node::custom::action::Print::text  [private]
```

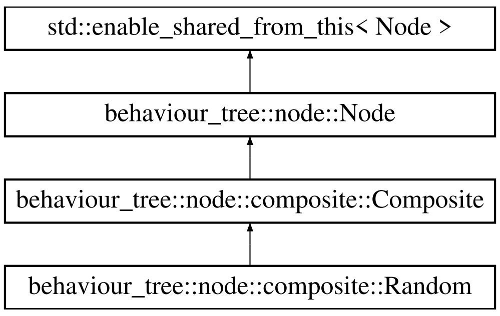
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/action/Print.hpp

7.23 behaviour_tree::node::composite::Random Class Reference

```
#include <Random.hpp>
```

Inheritance diagram for behaviour_tree::node::composite::Random:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::composite::Composite
behaviour_tree::node::composite::Random

Public Member Functions

- Random (const std::string &name, std::vector< std::shared_ptr< Node > > children)
- const CompositeType type () const final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context, const int start_index) final override
- const std::string toString () const final override

Additional Inherited Members

7.23.1 Constructor & Destructor Documentation

7.23.1.1 Random()

```
behaviour_tree::node::composite::Random::Random (
            const std::string & name,
            std::vector< std::shared_ptr< Node > > children )  [inline]
```

7.23.2 Member Function Documentation

7.23.2.1 run() [1/2]

```
const Status behaviour_tree::node::composite::Random::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.23.2.2 run() [2/2]

```
const Status behaviour_tree::node::composite::Random::run (
            const int tick_count,
            std::shared_ptr< Context > context,
            const int start_index )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::composite::Composite.

7.23.2.3 toString()

```
const std::string behaviour_tree::node::composite::Random::toString ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::Node.

7.23.2.4 type()

```
const CompositeType behaviour_tree::node::composite::Random::type ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::composite::Composite.

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

- behaviour_tree/include/behaviour_tree/node/composite/Random.hpp

7.24 utils::RawTypeNameFormat Struct Reference

```
#include <TypeName.hpp>
```

Public Attributes

- std::size_t leading_junk = 0
- std::size_t trailing_junk = 0

7.24.1 Member Data Documentation

7.24.1.1 leading_junk

```
std::size_t utils::RawTypeNameFormat::leading_junk = 0
```

7.24.1.2 trailing_junk

```
std::size_t utils::RawTypeNameFormat::trailing_junk = 0
```

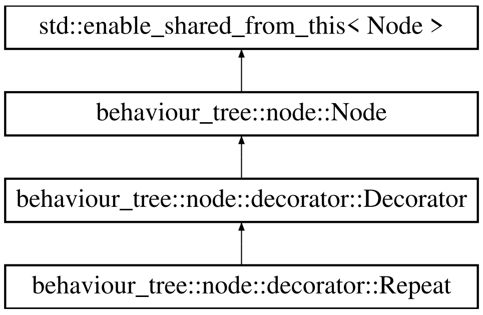
The documentation for this struct was generated from the following file:

- utils/include/utils/TypeName.hpp

7.25 behaviour_tree::node::decorator::Repeat Class Reference

```
#include <Repeat.hpp>
```

Inheritance diagram for behaviour_tree::node::decorator::Repeat:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::decorator::Decorator
behaviour_tree::node::decorator::Repeat

Public Member Functions

- Repeat (const std::string &name, std::shared_ptr< Node > child, const unsigned long amount, const bool break_on_fail)
- const DecoratorType type () const final override
- void start (std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override
- const unsigned long getAmount () const
- const bool getBreakOnFail () const
- const unsigned long getCount () const

Private Attributes

- const unsigned long amount
- const bool break_on_fail
- unsigned long count = 0

Additional Inherited Members

7.25.1 Constructor & Destructor Documentation

7.25.1.1 Repeat()

```
behaviour_tree::node::decorator::Repeat::Repeat (
            const std::string & name,
            std::shared_ptr< Node > child,
            const unsigned long amount,
            const bool break_on_fail ) [inline]
```

7.25.2 Member Function Documentation

7.25.2.1 getAmount()

```
const unsigned long behaviour_tree::node::decorator::Repeat::getAmount ( ) const  [inline]
```

7.25.2.2 getBreakOnFail()

```
const bool behaviour_tree::node::decorator::Repeat::getBreakOnFail ( ) const  [inline]
```

7.25.2.3 getCount()

```
const unsigned long behaviour_tree::node::decorator::Repeat::getCount ( ) const  [inline]
```

7.25.2.4 run()

```
const Status behaviour_tree::node::decorator::Repeat::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.25.2.5 start()

```
void behaviour_tree::node::decorator::Repeat::start (
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Reimplemented from behaviour_tree::node::Node.

7.25.2.6 toString()

```
const std::string behaviour_tree::node::decorator::Repeat::toString ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::Node.

7.25.2.7 type()

```
const DecoratorType behaviour_tree::node::decorator::Repeat::type ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::decorator::Decorator.

7.25.3 Member Data Documentation

7.25.3.1 amount

```
const unsigned long behaviour_tree::node::decorator::Repeat::amount  [private]
```

7.25.3.2 break_on_fail

```
const bool behaviour_tree::node::decorator::Repeat::break_on_fail  [private]
```

7.25.3.3 count

```
unsigned long behaviour_tree::node::decorator::Repeat::count = 0  [private]
```

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

- behaviour_tree/include/behaviour_tree/node/decorator/Repeat.hpp

7.26 behaviour_tree::Root Class Reference

```
#include <Root.hpp>
```

Public Member Functions

- Root (std::string id, std::shared_ptr< node::Node > child)
- const Status tick (const int tick_count, std::shared_ptr< Context > &context)
- const std::string toString () const
- const std::string & getId () const

Private Attributes

- std::string id
- std::shared_ptr< node::Node > child

7.26.1 Constructor & Destructor Documentation

7.26.1.1 Root()

```
behaviour_tree::Root::Root (
             std::string id,
             std::shared_ptr< node::Node > child )  [inline]
```

7.26.2 Member Function Documentation

7.26.2.1 getId()

```
const std::string & behaviour_tree::Root::getId ( ) const  [inline]
```

7.26.2.2 tick()

```
const Status behaviour_tree::Root::tick (
             const int tick_count,
             std::shared_ptr< Context > & context )  [inline]
```

7.26.2.3 toString()

```
const std::string behaviour_tree::Root::toString ( ) const  [inline]
```

7.26.3 Member Data Documentation

7.26.3.1 child

```
std::shared_ptr<node::Node> behaviour_tree::Root::child  [private]
```

7.26.3.2 id

```
std::string behaviour_tree::Root::id [private]
```

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

- behaviour_tree/include/behaviour_tree/Root.hpp

7.27 rplidar::RPLidar Class Reference

Class for communicating with RPLidar rangefinder scanners.

```
#include <RPLidar.h>
```

Public Member Functions

- RPLidar (const std::string &port, uint32_t baudrate, std::unique_ptr< serial::Serial > serial)

  Initialize RPLidar object for communicating with the sensor.
- ∼RPLidar ()
- void disconnect ()
- void set_motor_speed (int pwm)
- void start_motor ()

  Starts sensor motor.
- void stop_motor ()

  Stops sensor motor.
- tl::expected< DeviceInfo, std::string > get_info ()

  Get device information.
- tl::expected< HealthInfo, std::string > get_health ()

  Get device health state. When the core system detects some potential risk that may cause hardware failure in the future, the returned status value will be 'Warning'. But sensor can still work as normal. When sensor is in the Protection Stop state, the returned status value will be 'Error'. In case of warning or error statuses non-zero error code will be returned.
- void clean_input ()

  Clean input buffer by reading all available data.
- void stop ()

  Stops scanning process, disables laser diode and the measurement system, moves sensor to the idle state.
- tl::expected< nullptr_t, std::string > start (ScanType scanType=ScanType::NORMAL)

  Start the scanning process.
- void reset ()

  Resets sensor core, reverting it to a similar state as it has just been powered up.
- std::function< tl::expected< Measure, std::string >()> iter_measures (ScanType scanType=ScanType::NORMAL, int maxBufMeas=3000)
- std::function< std::vector< Measure >()> iter_scans (ScanType scanType=ScanType::NORMAL, int max↩BufMeas=3000, int minLen=5)

  Iterate over scans. Note that consumer must be fast enough, otherwise data will be accumulated inside buffer and consumer will get data with increasing lag.

Static Public Member Functions

- static tl::expected< std::unique_ptr< RPLidar >, std::string > create (const std::string &port, uint32_↩
  t baudrate=115200U) noexcept

Private Member Functions

- void _set_pwm (int pwm)
- void _send_payload_cmd (uint8_t cmd, const std::string &payload)

  Sends `cmd` command with `payload` to the sensor.
- void _send_cmd (uint8_t cmd)

  Sends `cmd` command to the sensor.
- tl::expected< std::tuple< uint8_t, bool, uint8_t >, std::string > _read_descriptor ()

  Reads descriptor packet.
- std::vector< uint8_t > _read_response (int dsize)

  Reads response packet with length of `dsize` bytes.
- std::string convertToHexString (uint8_t value)
- tl::expected< Measure, std::string > _process_scan (const std::vector< uint8_t > &raw)
- Measure _process_express_scan (std::unique_ptr< ExpressPacket > &data, float newAngle, int trame)

  Iterate over measures. Note that consumer must be fast enough, otherwise data will be accumulated inside buffer and consumer will get data with increasing lag.

Private Attributes

- std::unique_ptr< serial::Serial > _serial = nullptr
- std::string port
- uint32_t baudrate
- int _motor_speed = DEFAULT_MOTOR_PWM
- bool motor_running = false
- ScanInfo scanning = ScanInfo{false, 0, ScanType::NORMAL}
- int express_trame = 32
- std::unique_ptr< ExpressPacket > express_data = nullptr
- std::unique_ptr< ExpressPacket > express_old_data = nullptr

7.27.1 Detailed Description

Class for communicating with RPLidar rangefinder scanners.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 RPLidar()

```
rplidar::RPLidar::RPLidar (
            const std::string & port,
            uint32_t baudrate,
            std::unique_ptr< serial::Serial > serial )  [inline]
```

Initialize RPLidar object for communicating with the sensor.

Parameters

port	Serial port name to which sensor is connected
baudrate	Baudrate for serial connection (the default is 115200)

7.27.2.2 ∼RPLidar()

```
rplidar::RPLidar::∼RPLidar ( )  [inline]
```

7.27.3 Member Function Documentation

7.27.3.1 _process_express_scan()

```
Measure rplidar::RPLidar::_process_express_scan (
            std::unique_ptr< ExpressPacket > & data,
            float newAngle,
            int trame )  [inline], [private]
```

Iterate over measures. Note that consumer must be fast enough, otherwise data will be accumulated inside buffer and consumer will get data with increasing lag.

Parameters

scanType	
maxBufMeas	int or False if you want unlimited buffer Maximum number of bytes to be stored inside the buffer. Once numbe exceeds this limit buffer will be emptied out.

Returns

std::function<Measure()>

7.27.3.2 _process_scan()

```
tl::expected< Measure, std::string > rplidar::RPLidar::_process_scan (
            const std::vector< uint8_t > & raw )  [inline], [private]
```

7.27.3.3 _read_descriptor()

```
tl::expected< std::tuple< uint8_t, bool, uint8_t >, std::string > rplidar::RPLidar::_read_↩
descriptor ( )  [inline], [private]
```

Reads descriptor packet.

Returns

tl::expected<std::tuple<uint8_t, bool, uint8_t>, std::string>

7.27.3.4 _read_response()

```
std::vector< uint8_t > rplidar::RPLidar::_read_response (
             int dsize )  [inline], [private]
```

Reads response packet with length of `dsize` bytes.

Parameters

dsize	

Returns

std::vector<uint8_t>

7.27.3.5 _send_cmd()

```
void rplidar::RPLidar::_send_cmd (
             uint8_t cmd )  [inline], [private]
```

Sends `cmd` command to the sensor.

Parameters

cmd	

7.27.3.6 _send_payload_cmd()

```
void rplidar::RPLidar::_send_payload_cmd (
             uint8_t cmd,
             const std::string & payload )  [inline], [private]
```

Sends `cmd` command with `payload` to the sensor.

Parameters

cmd	
payload	

7.27.3.7 _set_pwm()

```
void rplidar::RPLidar::_set_pwm (
            int pwm )  [inline], [private]
```

7.27.3.8 clean_input()

```
void rplidar::RPLidar::clean_input ( )  [inline]
```

Clean input buffer by reading all available data.

7.27.3.9 convertToHexString()

```
std::string rplidar::RPLidar::convertToHexString (
            uint8_t value )  [inline], [private]
```

7.27.3.10 create()

```
static tl::expected< std::unique_ptr< RPLidar >, std::string > rplidar::RPLidar::create (
            const std::string & port,
            uint32_t baudrate = 115200U )  [inline], [static], [noexcept]
```

7.27.3.11 disconnect()

```
void rplidar::RPLidar::disconnect ( )  [inline]
```

7.27.3.12 get_health()

```
tl::expected< HealthInfo, std::string > rplidar::RPLidar::get_health ( )  [inline]
```

Get device health state. When the core system detects some potential risk that may cause hardware failure in the future, the returned status value will be 'Warning'. But sensor can still work as normal. When sensor is in the Protection Stop state, the returned status value will be 'Error'. In case of warning or error statuses non-zero error code will be returned.

Returns

tl::expected<HealthInfo, std::string>

7.27.3.13 get_info()

```
tl::expected< DeviceInfo, std::string > rplidar::RPLidar::get_info ( )  [inline]
```

Get device information.

Returns

tl::expected<DeviceInfo, std::string>

7.27.3.14 iter_measures()

```
std::function< tl::expected< Measure, std::string >()> rplidar::RPLidar::iter_measures (
            ScanType scanType = ScanType::NORMAL,
            int maxBufMeas = 3000 )  [inline]
```

7.27.3.15 iter_scans()

```
std::function< std::vector< Measure >()> rplidar::RPLidar::iter_scans (
            ScanType scanType = ScanType::NORMAL,
            int maxBufMeas = 3000,
            int minLen = 5 )  [inline]
```

Iterate over scans. Note that consumer must be fast enough, otherwise data will be accumulated inside buffer and consumer will get data with increasing lag.

Parameters

scanType	
maxBufMeas	Maximum number of measures to be stored inside the buffer. Once numbe exceeds this limit buffer will be emptied out.
minLen	Minimum number of measures in the scan for it to be yelded.

Returns

std::function<std::vector<Measure>()>

7.27.3.16 reset()

```
void rplidar::RPLidar::reset ( )  [inline]
```

Resets sensor core, reverting it to a similar state as it has just been powered up.

7.27.3.17 set_motor_speed()

```
void rplidar::RPLidar::set_motor_speed (
            int pwm )  [inline]
```

7.27.3.18 start()

```
tl::expected< nullptr_t, std::string > rplidar::RPLidar::start (
            ScanType scanType = ScanType::NORMAL )  [inline]
```

Start the scanning process.

Parameters

scanType	NORMAL, FORCE or EXPRESS

7.27.3.19 start_motor()

```
void rplidar::RPLidar::start_motor ( )  [inline]
```

Starts sensor motor.

7.27.3.20 stop()

```
void rplidar::RPLidar::stop ( )  [inline]
```

Stops scanning process, disables laser diode and the measurement system, moves sensor to the idle state.

7.27.3.21 stop_motor()

```
void rplidar::RPLidar::stop_motor ( )  [inline]
```

Stops sensor motor.

7.27.4 Member Data Documentation

7.27.4.1 _motor_speed

```
int rplidar::RPLidar::_motor_speed = DEFAULT_MOTOR_PWM  [private]
```

7.27.4.2 _serial

```
std::unique_ptr<serial::Serial> rplidar::RPLidar::_serial = nullptr  [private]
```

7.27.4.3 baudrate

```
uint32_t rplidar::RPLidar::baudrate  [private]
```

7.27.4.4 express_data

```
std::unique_ptr<ExpressPacket> rplidar::RPLidar::express_data = nullptr  [private]
```

7.27.4.5 express_old_data

```
std::unique_ptr<ExpressPacket> rplidar::RPLidar::express_old_data = nullptr  [private]
```

7.27.4.6 express_trame

```
int rplidar::RPLidar::express_trame = 32  [private]
```

7.27.4.7 motor_running

```
bool rplidar::RPLidar::motor_running = false  [private]
```

7.27.4.8 port

```
std::string rplidar::RPLidar::port  [private]
```

7.27.4.9 scanning

```
ScanInfo rplidar::RPLidar::scanning = ScanInfo{false, 0, ScanType::NORMAL}  [private]
```

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

- rplidar/include/RPLidar.h

7.28 rplidar::ScanInfo Struct Reference

```
#include <RPLidar.h>
```

Public Attributes

- int currently_scanning
- int dsize
- ScanType type

7.28.1 Member Data Documentation

7.28.1.1 currently_scanning

```
int rplidar::ScanInfo::currently_scanning
```

7.28.1.2 dsize

```
int rplidar::ScanInfo::dsize
```

7.28.1.3 type

```
ScanType rplidar::ScanInfo::type
```

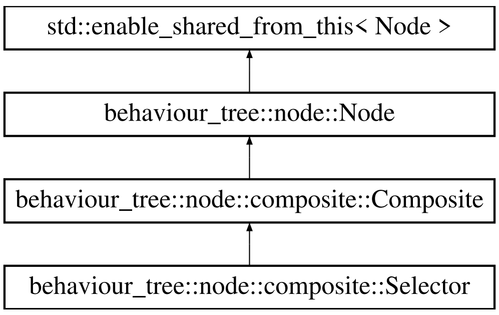
The documentation for this struct was generated from the following file:

- rplidar/include/RPLidar.h

7.29 behaviour_tree::node::composite::Selector Class Reference

```
#include <Selector.hpp>
```

Inheritance diagram for behaviour_tree::node::composite::Selector:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::composite::Composite
behaviour_tree::node::composite::Selector

Public Member Functions

- Selector (const std::string &name, std::vector< std::shared_ptr< Node > > children)
- const CompositeType type () const final override
- void start (std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context, const int start_index) final override
- const std::string toString () const final override

Private Attributes

- int previous_start_index = -1

Additional Inherited Members

7.29.1 Constructor & Destructor Documentation

7.29.1.1 Selector()

```
behaviour_tree::node::composite::Selector::Selector (
            const std::string & name,
            std::vector< std::shared_ptr< Node > > children )  [inline]
```

7.29.2 Member Function Documentation

7.29.2.1 run() [1/2]

```
const Status behaviour_tree::node::composite::Selector::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.29.2.2 run() [2/2]

```
const Status behaviour_tree::node::composite::Selector::run (
            const int tick_count,
            std::shared_ptr< Context > context,
            const int start_index )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::composite::Composite.

7.29.2.3 start()

```
void behaviour_tree::node::composite::Selector::start (
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Reimplemented from behaviour_tree::node::Node.

7.29.2.4 toString()

```
const std::string behaviour_tree::node::composite::Selector::toString ( ) const  [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.29.2.5 type()

```
const CompositeType behaviour_tree::node::composite::Selector::type ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::composite::Composite.

7.29.3 Member Data Documentation

7.29.3.1 previous_start_index

```
int behaviour_tree::node::composite::Selector::previous_start_index = -1  [private]
```

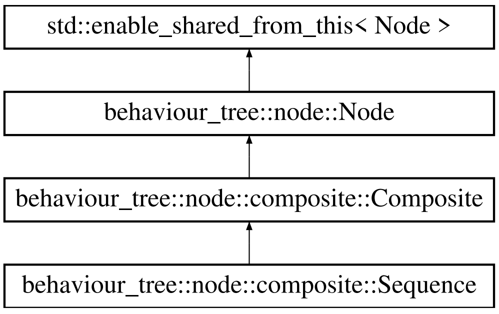
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/composite/Selector.hpp

7.30 behaviour_tree::node::composite::Sequence Class Reference

```
#include <Sequence.hpp>
```

Inheritance diagram for behaviour_tree::node::composite::Sequence:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::composite::Composite
behaviour_tree::node::composite::Sequence

Public Member Functions

- Sequence (const std::string &name, std::vector< std::shared_ptr< Node > > children)
- const CompositeType type () const final override
- void start (std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const Status run (const int tick_count, std::shared_ptr< Context > context, const int start_index) final override
- const std::string toString () const final override

Private Attributes

- int previous_start_index = -1

Additional Inherited Members

7.30.1 Constructor & Destructor Documentation

7.30.1.1 Sequence()

```
behaviour_tree::node::composite::Sequence::Sequence (
            const std::string & name,
            std::vector< std::shared_ptr< Node > > children )  [inline]
```

7.30.2 Member Function Documentation

7.30.2.1 run() [1/2]

```
const Status behaviour_tree::node::composite::Sequence::run (
            const int tick_count,
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.30.2.2 run() [2/2]

```
const Status behaviour_tree::node::composite::Sequence::run (
            const int tick_count,
            std::shared_ptr< Context > context,
            const int start_index )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::composite::Composite.

7.30.2.3 start()

```
void behaviour_tree::node::composite::Sequence::start (
            std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Reimplemented from behaviour_tree::node::Node.

7.30.2.4 toString()

```
const std::string behaviour_tree::node::composite::Sequence::toString ( ) const  [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.30.2.5 type()

```
const CompositeType behaviour_tree::node::composite::Sequence::type ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::composite::Composite.

7.30.3 Member Data Documentation

7.30.3.1 previous_start_index

```
int behaviour_tree::node::composite::Sequence::previous_start_index = -1  [private]
```

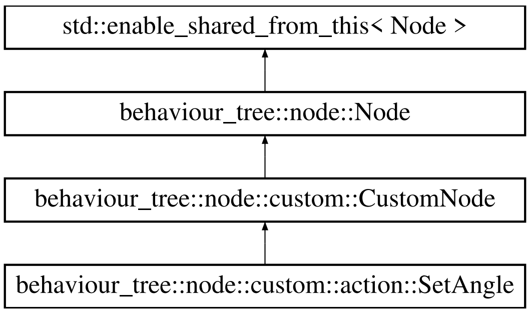
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/composite/Sequence.hpp

7.31 behaviour_tree::node::custom::action::SetAngle Class Reference

```
#include <SetAngle.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::action::SetAngle:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::SetAngle

Public Member Functions

- SetAngle (const std::string &name, const ServoType servo_type, const int angle)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override
- const ServoType getServoType () const
- const int getAngle () const

Static Public Member Functions

- static const tl::expected< std::shared_ptr< SetAngle >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const int angle
- const ServoType servo_type

Additional Inherited Members

7.31.1 Constructor & Destructor Documentation

7.31.1.1 SetAngle()

```
behaviour_tree::node::custom::action::SetAngle::SetAngle (
            const std::string & name,
            const ServoType servo_type,
            const int angle )  [inline]
```

7.31.2 Member Function Documentation

7.31.2.1 getAngle()

```
const int behaviour_tree::node::custom::action::SetAngle::getAngle ( ) const  [inline]
```

7.31.2.2 getServoType()

```
const ServoType behaviour_tree::node::custom::action::SetAngle::getServoType ( ) const  [inline]
```

7.31.2.3 parse()

```
static const tl::expected< std::shared_ptr< SetAngle >, std::string > behaviour_tree::node↩
::custom::action::SetAngle::parse (
            const pugi::xml_node & node,
            const int index,
            const std::string & name_attribute )  [inline], [static]
```

7.31.2.4 run()

```
const Status behaviour_tree::node::custom::action::SetAngle::run (
            const int tick_count,
            std::shared_ptr< Context > context ) [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.31.2.5 toString()

```
const std::string behaviour_tree::node::custom::action::SetAngle::toString ( ) const [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.31.3 Member Data Documentation

7.31.3.1 angle

```
const int behaviour_tree::node::custom::action::SetAngle::angle [private]
```

7.31.3.2 servo_type

```
const ServoType behaviour_tree::node::custom::action::SetAngle::servo_type [private]
```

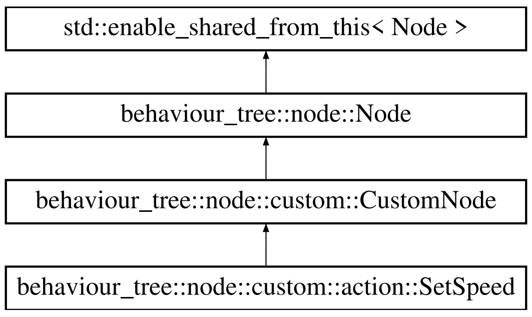
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/action/SetAngle.hpp

7.32 behaviour_tree::node::custom::action::SetSpeed Class Reference

```
#include <SetSpeed.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::action::SetSpeed:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::SetSpeed

Public Member Functions

- SetSpeed (const std::string &name, const WheelType wheel_type, const int speed)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override
- const WheelType getWheelType () const
- const int getSpeed () const

Static Public Member Functions

- static const tl::expected< std::shared_ptr< SetSpeed >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const int speed
- const WheelType wheel_type

Additional Inherited Members

7.32.1 Constructor & Destructor Documentation

7.32.1.1 SetSpeed()

```
behaviour_tree::node::custom::action::SetSpeed::SetSpeed (
            const std::string & name,
            const WheelType wheel_type,
            const int speed )  [inline]
```

7.32.2 Member Function Documentation

7.32.2.1 getSpeed()

```
const int behaviour_tree::node::custom::action::SetSpeed::getSpeed ( ) const  [inline]
```

7.32.2.2 getWheelType()

```
const WheelType behaviour_tree::node::custom::action::SetSpeed::getWheelType ( ) const  [inline]
```

7.32.2.3 parse()

```
static const tl::expected< std::shared_ptr< SetSpeed >, std::string > behaviour_tree::node↩
::custom::action::SetSpeed::parse (
             const pugi::xml_node & node,
             const int index,
             const std::string & name_attribute )  [inline], [static]
```

7.32.2.4 run()

```
const Status behaviour_tree::node::custom::action::SetSpeed::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.32.2.5 toString()

```
const std::string behaviour_tree::node::custom::action::SetSpeed::toString ( ) const  [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.32.3 Member Data Documentation

7.32.3.1 speed

```
const int behaviour_tree::node::custom::action::SetSpeed::speed  [private]
```

7.32.3.2 wheel_type

```
const WheelType behaviour_tree::node::custom::action::SetSpeed::wheel_type  [private]
```

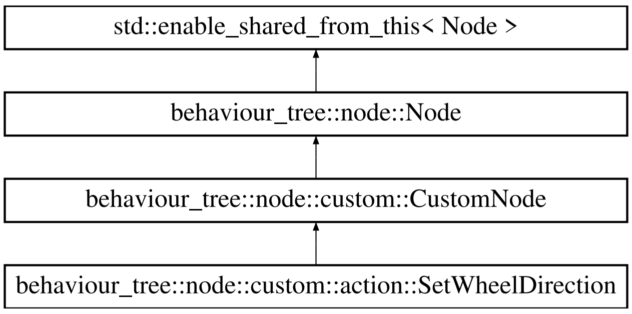
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/action/SetSpeed.hpp

7.33 behaviour_tree::node::custom::action::SetWheelDirection Class Reference

`#include <SetWheelDirection.hpp>`

Inheritance diagram for behaviour_tree::node::custom::action::SetWheelDirection:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::SetWheelDirection

Public Member Functions

- SetWheelDirection (const std::string &name, const WheelType wheel_type, const DirectionType direction_type)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override
- const DirectionType getDirectionType () const
- const WheelType getWheelType () const

Static Public Member Functions

- static const tl::expected< std::shared_ptr< SetWheelDirection >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const DirectionType direction_type
- const WheelType wheel_type

Additional Inherited Members

7.33.1 Constructor & Destructor Documentation

7.33.1.1 SetWheelDirection()

```
behaviour_tree::node::custom::action::SetWheelDirection::SetWheelDirection (
            const std::string & name,
            const WheelType wheel_type,
            const DirectionType direction_type )  [inline]
```

7.33.2 Member Function Documentation

7.33.2.1 getDirectionType()

```
const DirectionType behaviour_tree::node::custom::action::SetWheelDirection::getDirectionType
( ) const  [inline]
```

7.33.2.2 getWheelType()

```
const WheelType behaviour_tree::node::custom::action::SetWheelDirection::getWheelType ( )
const  [inline]
```

7.33.2.3 parse()

```
static const tl::expected< std::shared_ptr< SetWheelDirection >, std::string > behaviour_↩
tree::node::custom::action::SetWheelDirection::parse (
             const pugi::xml_node & node,
             const int index,
             const std::string & name_attribute )  [inline], [static]
```

7.33.2.4 run()

```
const Status behaviour_tree::node::custom::action::SetWheelDirection::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.33.2.5 toString()

```
const std::string behaviour_tree::node::custom::action::SetWheelDirection::toString ( ) const
[inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.33.3 Member Data Documentation

7.33.3.1 direction_type

```
const DirectionType behaviour_tree::node::custom::action::SetWheelDirection::direction_type
[private]
```

7.33.3.2 wheel_type

```
const WheelType behaviour_tree::node::custom::action::SetWheelDirection::wheel_type  [private]
```

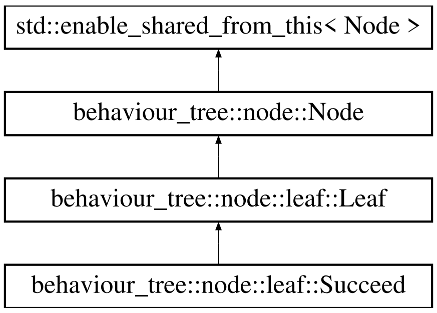
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/action/SetWheelDirection.hpp

7.34 behaviour_tree::node::leaf::Succeed Class Reference

```
#include <Succeed.hpp>
```

Inheritance diagram for behaviour_tree::node::leaf::Succeed:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::leaf::Leaf
behaviour_tree::node::leaf::Succeed

Public Member Functions

- Succeed (const std::string &name)
- const LeafType type () const final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override

Additional Inherited Members

7.34.1 Constructor & Destructor Documentation

7.34.1.1 Succeed()

```
behaviour_tree::node::leaf::Succeed::Succeed (
             const std::string & name )  [inline]
```

7.34.2 Member Function Documentation

7.34.2.1 run()

```
const Status behaviour_tree::node::leaf::Succeed::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.34.2.2 toString()

```
const std::string behaviour_tree::node::leaf::Succeed::toString ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::Node.

7.34.2.3 type()

```
const LeafType behaviour_tree::node::leaf::Succeed::type ( ) const  [inline], [final], [override],
[virtual]
```

Implements behaviour_tree::node::leaf::Leaf.

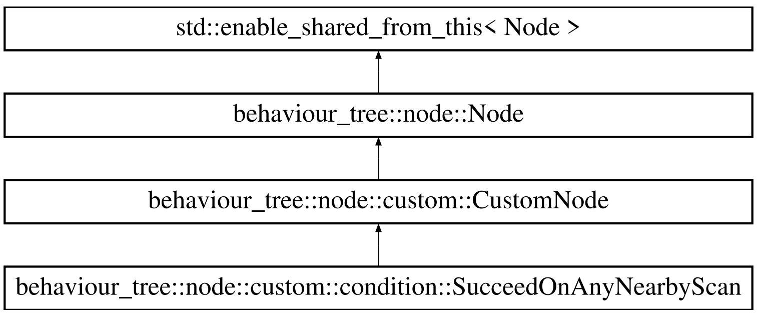
The documentation for this class was generated from the following file:

- behaviour_tree/include/behaviour_tree/node/leaf/Succeed.hpp

7.35 behaviour_tree::node::custom::condition::SucceedOnAnyNearby↩ Scan Class Reference

```
#include <SucceedOnAnyNearbyScan.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan

Public Member Functions

- SucceedOnAnyNearbyScan (const std::string &name, const int min_angle, const int max_angle, const double cm)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const int getMinAngle () const
- const int getMaxAngle () const
- const double getMinimumDistanceUnit () const
- const double getCentimeters () const
- const std::string toString () const final override

Static Public Member Functions

- static const tl::expected< std::shared_ptr< SucceedOnAnyNearbyScan >, std::string > parse (const pugi↩ ::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const int min_angle
- const int max_angle
- const double cm
- const double minimum_distance_unit

Additional Inherited Members

7.35.1 Constructor & Destructor Documentation

7.35.1.1 SucceedOnAnyNearbyScan()

```
behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::SucceedOnAnyNearbyScan (
             const std::string & name,
             const int min_angle,
             const int max_angle,
             const double cm )  [inline]
```

7.35.2 Member Function Documentation

7.35.2.1 getCentimeters()

```
const double behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::getCentimeters (
) const  [inline]
```

7.35.2.2 getMaxAngle()

```
const int behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::getMaxAngle ( )
const  [inline]
```

7.35.2.3 getMinAngle()

```
const int behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::getMinAngle ( )
const  [inline]
```

7.35.2.4 getMinimumDistanceUnit()

```
const double behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::getMinimum↩
DistanceUnit ( ) const  [inline]
```

7.35.2.5 parse()

```
static const tl::expected< std::shared_ptr< SucceedOnAnyNearbyScan >, std::string > behaviour↩
_tree::node::custom::condition::SucceedOnAnyNearbyScan::parse (
             const pugi::xml_node & node,
             const int index,
             const std::string & name_attribute )  [inline], [static]
```

7.35.2.6 run()

```
const Status behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.35.2.7 toString()

```
const std::string behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::toString (
) const  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.35.3 Member Data Documentation

7.35.3.1 cm

```
const double behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::cm  [private]
```

7.35.3.2 max_angle

```
const int behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::max_angle  [private]
```

7.35.3.3 min_angle

```
const int behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::min_angle  [private]
```

7.35.3.4 minimum_distance_unit

```
const double behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan::minimum_distance←
_unit  [private]
```

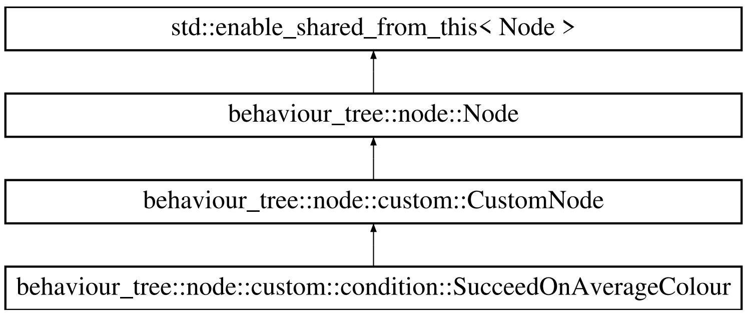
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/condition/SucceedOnAnyNearbyScan.hpp

7.36 behaviour_tree::node::custom::condition::SucceedOnAverage← Colour Class Reference

```
#include <SucceedOnAverageColour.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::condition::SucceedOnAverageColour:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::condition::SucceedOnAverageColour

Public Member Functions

- SucceedOnAverageColour (const std::string &name, const std::string hex_colour, const double tolerance)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- int calculateColorDifference (const std::string &color1, const std::string &color2)
- cv::Scalar hexToRGB (const std::string &hexColor)
- int calculateMaxColorDifference (double percentage)
- const std::string getHexColour () const
- const double getTolerance () const
- const std::string toString () const final override

Static Public Member Functions

- static const tl::expected< std::shared_ptr< SucceedOnAverageColour >, std::string > parse (const pugi↩ ::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const std::string hex_colour
- const double tolerance

Additional Inherited Members

7.36.1 Constructor & Destructor Documentation

7.36.1.1 SucceedOnAverageColour()

```
behaviour_tree::node::custom::condition::SucceedOnAverageColour::SucceedOnAverageColour (
            const std::string & name,
            const std::string hex_colour,
            const double tolerance )  [inline]
```

7.36.2 Member Function Documentation

7.36.2.1 calculateColorDifference()

```
int behaviour_tree::node::custom::condition::SucceedOnAverageColour::calculateColorDifference
(
            const std::string & color1,
            const std::string & color2 )  [inline]
```

7.36.2.2 calculateMaxColorDifference()

```
int behaviour_tree::node::custom::condition::SucceedOnAverageColour::calculateMaxColorDifference
(
             double percentage )  [inline]
```

7.36.2.3 getHexColour()

```
const std::string behaviour_tree::node::custom::condition::SucceedOnAverageColour::getHex←
Colour ( ) const  [inline]
```

7.36.2.4 getTolerance()

```
const double behaviour_tree::node::custom::condition::SucceedOnAverageColour::getTolerance ( )
const  [inline]
```

7.36.2.5 hexToRGB()

```
cv::Scalar behaviour_tree::node::custom::condition::SucceedOnAverageColour::hexToRGB (
             const std::string & hexColor )  [inline]
```

7.36.2.6 parse()

```
static const tl::expected< std::shared_ptr< SucceedOnAverageColour >, std::string > behaviour←
_tree::node::custom::condition::SucceedOnAverageColour::parse (
             const pugi::xml_node & node,
             const int index,
             const std::string & name_attribute )  [inline], [static]
```

7.36.2.7 run()

```
const Status behaviour_tree::node::custom::condition::SucceedOnAverageColour::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.36.2.8 toString()

```
const std::string behaviour_tree::node::custom::condition::SucceedOnAverageColour::toString (
) const  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.36.3 Member Data Documentation

7.36.3.1 hex_colour

```
const std::string behaviour_tree::node::custom::condition::SucceedOnAverageColour::hex_colour
[private]
```

7.36.3.2 tolerance

```
const double behaviour_tree::node::custom::condition::SucceedOnAverageColour::tolerance  [private]
```

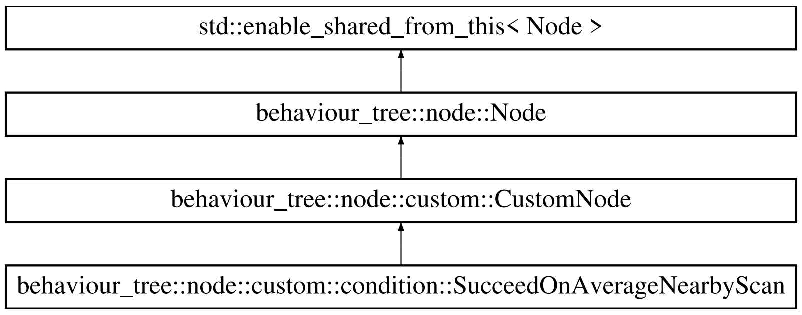
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/condition/SucceedOnAverageColour.hpp

7.37 behaviour_tree::node::custom::condition::SucceedOnAverage←NearbyScan Class Reference

```
#include <SucceedOnAverageNearbyScan.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan

Public Member Functions

- SucceedOnAverageNearbyScan (const std::string &name, const int min_angle, const int max_angle, const double cm, const int smallest_measure_amount_used)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const int getMinAngle () const
- const int getMaxAngle () const
- const double getAverageDistanceUnit () const
- const double getCentimeters () const
- const int getSmallestMeasureAmountUsed () const
- const std::string toString () const final override

Static Public Member Functions

- static const tl::expected< std::shared_ptr< SucceedOnAverageNearbyScan >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const int min_angle
- const int max_angle
- const double cm
- const int smallest_measure_amount_used
- const double average_distance_unit

Additional Inherited Members

7.37.1 Constructor & Destructor Documentation

7.37.1.1 SucceedOnAverageNearbyScan()

```
behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::SucceedOnAverageNearby←
Scan (
            const std::string & name,
            const int min_angle,
            const int max_angle,
            const double cm,
            const int smallest_measure_amount_used )  [inline]
```

7.37.2 Member Function Documentation

7.37.2.1 getAverageDistanceUnit()

```
const double behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::getAverage←
DistanceUnit ( ) const  [inline]
```

7.37.2.2 getCentimeters()

```
const double behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::getCentimeters
( ) const  [inline]
```

7.37.2.3 getMaxAngle()

```
const int behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::getMaxAngle ( )
const  [inline]
```

7.37.2.4 getMinAngle()

```
const int behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::getMinAngle ( )
const  [inline]
```

7.37.2.5 getSmallestMeasureAmountUsed()

```
const int behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::getSmallest←
MeasureAmountUsed ( ) const  [inline]
```

7.37.2.6 parse()

```
static const tl::expected< std::shared_ptr< SucceedOnAverageNearbyScan >, std::string > behaviour←
_tree::node::custom::condition::SucceedOnAverageNearbyScan::parse (
             const pugi::xml_node & node,
             const int index,
             const std::string & name_attribute )  [inline], [static]
```

7.37.2.7 run()

```
const Status behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.37.2.8 toString()

```
const std::string behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::to↩
String ( ) const  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.37.3 Member Data Documentation

7.37.3.1 average_distance_unit

```
const double behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::average_↩
distance_unit  [private]
```

7.37.3.2 cm

```
const double behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::cm  [private]
```

7.37.3.3 max_angle

```
const int behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::max_angle  [private]
```

7.37.3.4 min_angle

```
const int behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::min_angle  [private]
```

7.37.3.5 smallest_measure_amount_used

```
const int behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan::smallest_↩
measure_amount_used [private]
```

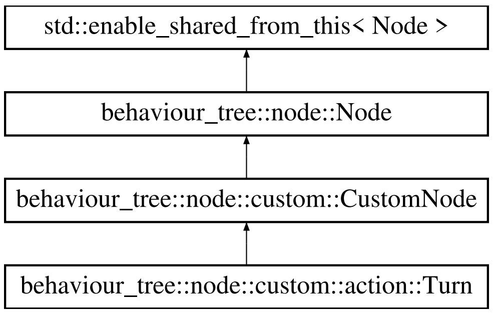
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/condition/SucceedOnAverageNearbyScan.hpp

7.38 behaviour_tree::node::custom::action::Turn Class Reference

```
#include <Turn.hpp>
```

Inheritance diagram for behaviour_tree::node::custom::action::Turn:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::custom::CustomNode
behaviour_tree::node::custom::action::Turn

Public Member Functions

- Turn (const std::string &name, const int angle)
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string toString () const final override
- const int getAngle () const

Static Public Member Functions

- static const tl::expected< std::shared_ptr< Turn >, std::string > parse (const pugi::xml_node &node, const int index, const std::string &name_attribute)

Private Attributes

- const int angle

Additional Inherited Members

7.38.1 Constructor & Destructor Documentation

7.38.1.1 Turn()

```
behaviour_tree::node::custom::action::Turn::Turn (
            const std::string & name,
            const int angle ) [inline]
```

7.38.2 Member Function Documentation

7.38.2.1 getAngle()

```
const int behaviour_tree::node::custom::action::Turn::getAngle ( ) const [inline]
```

7.38.2.2 parse()

```
static const tl::expected< std::shared_ptr< Turn >, std::string > behaviour_tree::node↩
::custom::action::Turn::parse (
            const pugi::xml_node & node,
            const int index,
            const std::string & name_attribute ) [inline], [static]
```

7.38.2.3 run()

```
const Status behaviour_tree::node::custom::action::Turn::run (
            const int tick_count,
            std::shared_ptr< Context > context ) [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.38.2.4 toString()

```
const std::string behaviour_tree::node::custom::action::Turn::toString ( ) const [inline],
[final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.38.3 Member Data Documentation

7.38.3.1 angle

```
const int behaviour_tree::node::custom::action::Turn::angle  [private]
```

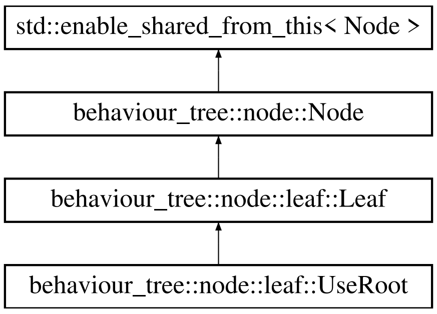
The documentation for this class was generated from the following file:

- behaviour_tree_car/include/behaviour_tree/node/custom/action/Turn.hpp

7.39 behaviour_tree::node::leaf::UseRoot Class Reference

```
#include <UseRoot.hpp>
```

Inheritance diagram for behaviour_tree::node::leaf::UseRoot:


std::enable_shared_from_this< Node >
behaviour_tree::node::Node
behaviour_tree::node::leaf::Leaf
behaviour_tree::node::leaf::UseRoot

Public Member Functions

- UseRoot (const std::string &name, const std::string &id)
- const LeafType type () const final override
- const Status run (const int tick_count, std::shared_ptr< Context > context) final override
- const std::string & getId () const
- const std::string toString () const final override

Private Attributes

- const std::string id

Additional Inherited Members

7.39.1 Constructor & Destructor Documentation

7.39.1.1 UseRoot()

```
behaviour_tree::node::leaf::UseRoot::UseRoot (
            const std::string & name,
            const std::string & id )  [inline]
```

7.39.2 Member Function Documentation

7.39.2.1 getId()

```
const std::string & behaviour_tree::node::leaf::UseRoot::getId ( ) const  [inline]
```

7.39.2.2 run()

```
const Status behaviour_tree::node::leaf::UseRoot::run (
             const int tick_count,
             std::shared_ptr< Context > context )  [inline], [final], [override], [virtual]
```

Implements behaviour_tree::node::Node.

7.39.2.3 toString()

```
const std::string behaviour_tree::node::leaf::UseRoot::toString ( ) const  [inline], [final],
[override], [virtual]
```

Implements behaviour_tree::node::Node.

7.39.2.4 type()

```
const LeafType behaviour_tree::node::leaf::UseRoot::type ( ) const  [inline], [final], [override],
[virtual]
```

Implements behaviour_tree::node::leaf::Leaf.

7.39.3 Member Data Documentation

7.39.3.1 id

```
const std::string behaviour_tree::node::leaf::UseRoot::id  [private]
```

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

- behaviour_tree/include/behaviour_tree/node/leaf/UseRoot.hpp

Chapter 8

File Documentation

8.1 behaviour_tree/include/behaviour_tree/BehaviourTree.hpp File Reference

`#include "Root.hpp"`

Classes

- class behaviour_tree::BehaviourTree

Namespaces

- namespace behaviour_tree

8.2 BehaviourTree.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_HPP
#define BEHAVIOUR_TREE_HPP

#pragma once

/*
XML format inspired by: https://github.com/telcy/modular-behavior-tree &
     https://github.com/Defaultio/BehaviorTree3
*/

#include "Root.hpp"

namespace behaviour_tree
{
    class BehaviourTree
    {
    public:
        BehaviourTree(unsigned int cycle_limit, std::vector<std::shared_ptr<Root» roots) :
    cycle_limit(cycle_limit), roots(std::move(roots))
        {
        }

        void start()
        {
            for (auto& root : this->roots)
            {
                if (root->getId() == "Main")
                {
                    this->root_to_use = root;
                    return;
                }
            }
            this->root_to_use = this->roots[0];
        }

        Status tick(const int tick_count, std::shared_ptr<Context> context)
        {
            const Status status = this->root_to_use->tick(tick_count, context);
            if (status != Status::Running)
            {
                this->cycles++;
            }
            return status;
        }

        Status UseRoot(const int tick_count, std::shared_ptr<Context> context, const std::string& id) {
            for (auto& root : this->roots)
            {
                if (root->getId() == id)
                {
                    return root->tick(tick_count, context);
                }
            }
            return Status::Failure;
        }

        bool canRun() const {
            if (this->cycle_limit <= 0)
            {
                return true;
            }
            return this->cycles < this->cycle_limit;
        }

        void incrementCycle() {
            this->cycles++;
        }

        void resetCycles() {
            this->cycles = 0;
        }

        const std::string toString() const {
            std::string out;
            for (auto& root : this->roots)
            {
                out += root->toString();
            }
            return fmt::format(R"(<BehaviourTree cycle_limit='{}'>{}</BehaviourTree>)",
    this->cycle_limit, out);
        }

    private:
        int cycles = 0;
        std::shared_ptr<Root> root_to_use;

        const std::vector<std::shared_ptr<Root» roots;
        const unsigned int cycle_limit;
    };
}

#endif
```

8.3 behaviour_tree/include/behaviour_tree/BehaviourTreeParser.hpp File Reference

```
#include <filesystem>
#include <string>
#include <numeric>
#include <tl/expected.hpp>
#include <pugixml.hpp>
#include <fmt/format.h>
```

```
#include "utils/Utility.hpp"
#include "BehaviourTree.hpp"
#include "Root.hpp"
#include "node/Node.hpp"
#include "node/composite/Composite.hpp"
#include "node/composite/Sequence.hpp"
#include "node/composite/Selector.hpp"
#include "node/composite/Random.hpp"
#include "node/decorator/Decorator.hpp"
#include "node/decorator/Repeat.hpp"
#include "node/decorator/Invert.hpp"
#include "node/leaf/Leaf.hpp"
#include "node/leaf/Succeed.hpp"
#include "node/leaf/Fail.hpp"
#include "node/leaf/UseRoot.hpp"
#include "node/blackboard/ChangeInteger.hpp"
#include "node/blackboard/IntegerCondition.hpp"
#include "node/custom/CustomNode.hpp"
#include "node/custom/CustomNodeParser.hpp"
```

Classes

- class behaviour_tree::BehaviourTreeParser

Namespaces

- namespace behaviour_tree

8.4 BehaviourTreeParser.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_PARSER_HPP
#define BEHAVIOUR_TREE_PARSER_HPP

#pragma once

#include <filesystem>
#include <string>
#include <numeric>

#include <tl/expected.hpp>
#include <pugixml.hpp>
#include <fmt/format.h>

#include "utils/Utility.hpp"

#include "BehaviourTree.hpp"
#include "Root.hpp"
#include "node/Node.hpp"

#include "node/composite/Composite.hpp"
#include "node/composite/Sequence.hpp"
#include "node/composite/Selector.hpp"
#include "node/composite/Random.hpp"

#include "node/decorator/Decorator.hpp"
#include "node/decorator/Repeat.hpp"
#include "node/decorator/Invert.hpp"

#include "node/leaf/Leaf.hpp"
#include "node/leaf/Succeed.hpp"
#include "node/leaf/Fail.hpp"
#include "node/leaf/UseRoot.hpp"
```

```

#include "node/blackboard/ChangeInteger.hpp"
#include "node/blackboard/IntegerCondition.hpp"

#include "node/custom/CustomNode.hpp"

#include "node/custom/CustomNodeParser.hpp"

using namespace behaviour_tree::node;
using namespace behaviour_tree::node::composite;
using namespace behaviour_tree::node::decorator;
using namespace behaviour_tree::node::leaf;
using namespace behaviour_tree::node::blackboard;
using namespace behaviour_tree::node::custom;

namespace behaviour_tree
{
    static constexpr int STARTING_INDEX = 1;

    class BehaviourTreeParser
    {
    public:
        BehaviourTreeParser(BehaviourTreeParser const&) = delete;
        void operator=(BehaviourTreeParser const&) = delete;

        static BehaviourTreeParser& instance()
        {
            static BehaviourTreeParser instance;
            return instance;
        }

        void setCustomNodeParser(std::shared_ptr<CustomNodeParser> custom_node_parser)
        {
            this->custom_node_parser = std::move(custom_node_parser);
        }

        tl::expected<std::shared_ptr<BehaviourTree>, std::string> parseXML(const std::string& xml)
        {
            pugi::xml_document doc;
            pugi::xml_parse_result result = doc.load_string(xml.c_str());
            if (!result)
            {
                return tl::make_unexpected("XML parse error: " + std::string(result.description()));
            }
            return parse(doc);
        }

        tl::expected<std::shared_ptr<BehaviourTree>, std::string> parseFileXML(const std::string&
    file_path)
        {
            if (std::filesystem::exists(file_path) == false)
            {
                return tl::make_unexpected("File [" + file_path + "] does not exist");
            }
            pugi::xml_document doc;
            pugi::xml_parse_result result = doc.load_file(file_path.c_str());
            if (!result)
            {
                return tl::make_unexpected("XML [" + file_path + "] parsed with errors");
            }
            return parse(doc);
        }

    private:
        BehaviourTreeParser() {}

        std::shared_ptr<CustomNodeParser> custom_node_parser;

         tl::expected<std::shared_ptr<BehaviourTree>, std::string> parse(pugi::xml_document& doc)
         {
             pugi::xml_node behaviour_tree_node = doc.child("BehaviourTree");
             if (!behaviour_tree_node)
             {
                 return tl::make_unexpected(R"(No "BehaviourTree" node found)");
             }
             std::vector<std::shared_ptr<Root» roots;
             for (pugi::xml_node node = behaviour_tree_node.child("Root"); node; node =
    node.next_sibling("Root"))
             {
                 const auto maybe_root = parseRoot(node, STARTING_INDEX + roots.size());
                 if (!maybe_root.has_value())
                 {
                     return tl::make_unexpected(maybe_root.error());
                 }
                 roots.push_back(std::move(maybe_root.value()));
             }
             if (roots.size() <= 0)
```

```cpp
            {
                return tl::make_unexpected(R"(No "Root" nodes found in BehaviourTree)");
            }
            const int cycle_limit = behaviour_tree_node.attribute("cycle_limit").as_int(0);
            return std::make_shared<BehaviourTree>(
                BehaviourTree(cycle_limit, std::move(roots)));
        }

    private:
        tl::expected<std::shared_ptr<Root>, std::string> parseRoot(const pugi::xml_node& node, const int index)
        {
            int child_count = 0;
            for (pugi::xml_node child = node.first_child(); child; child = child.next_sibling())
                ++child_count;
            const std::string id = node.attribute("id").as_string();
            if (child_count == 0)
            {
                return std::make_shared<Root>(
                    Root(id, nullptr));
            }
            if (child_count != 1)
            {
                return tl::make_unexpected(fmt::format(R"(Root node must have only at most one child | Root:['{}',{}])", id, index));
            }
            const pugi::xml_node child = node.first_child();
            const auto maybe_child_node = parseChild(child, STARTING_INDEX);
            if (!maybe_child_node.has_value())
            {
                return tl::make_unexpected(fmt::format(R"({}<-Root:['{}',{}])", maybe_child_node.error(), id, index));
            }
            return std::make_shared<Root>(
                Root(id, std::move(maybe_child_node.value())));
        }

        tl::expected<std::shared_ptr<node::Node>, std::string> parseChild(const pugi::xml_node& node, const int index)
        {
            const std::string name_attribute = node.attribute("name").as_string();
            const std::string& node_name = node.name();
            switch (utils::hash(node_name))
            {
#pragma region Composite Node
            case utils::hash("Sequence"):
            {
                return parseComposite(node, index, CompositeType::Sequence);
            }
            case utils::hash("Selector"):
            {
                return parseComposite(node, index, CompositeType::Selector);
            }
            case utils::hash("Random"):
            {
                return parseComposite(node, index, CompositeType::Random);
            }
#pragma endregion
#pragma region Decorator Node
            case utils::hash("Repeat"):
            {
                return parseDecorator(node, index, DecoratorType::Repeat);
            }
            case utils::hash("Invert"):
            {
                return parseDecorator(node, index, DecoratorType::Invert);
            }
#pragma endregion
#pragma region Leaf Node
            case utils::hash("Succeed"):
            {
                return std::make_shared<Succeed>(Succeed(name_attribute));
            }
            case utils::hash("Fail"):
            {
                return std::make_shared<Fail>(Fail(name_attribute));
            }
            case utils::hash("UseRoot"):
            {
                return std::make_shared<UseRoot>(
                    UseRoot(
                        name_attribute,
                        node.attribute("id").as_string()));
            }
#pragma endregion
#pragma region Blackboard Leaf Node
            case utils::hash("Blackboard:ChangeInteger"):
```

```
            {
                return parseBlackboardNode(node, index, BlackboardType::ChangeInteger);
            }
            case utils::hash("Blackboard:IntegerCondition"):
            {
                return parseBlackboardNode(node, index, BlackboardType::IntegerCondition);
            }
#pragma endregion
            default:
            {
                return this->parseCustomNode(node, index);
            }
            }
        }

        tl::expected<std::shared_ptr<node::decorator::Decorator>, std::string> parseDecorator(const pugi::xml_node& node, const int index, const DecoratorType decorator_type)
        {
            const std::string name_attribute = node.attribute("name").as_string();
            const pugi::xml_node child = node.first_child();
            if (!child)
            {
                return tl::make_unexpected(fmt::format(R"(Decorator node must have a child | {}:['{}',{}])", std::string(node.name()), name_attribute, index));
            }
            const auto maybe_child_node = parseChild(child, STARTING_INDEX);
            if (!maybe_child_node.has_value())
            {
                return tl::make_unexpected(maybe_child_node.error());
            }
            switch (decorator_type)
            {
            case DecoratorType::Invert:
            {
                return std::make_shared<Invert>(
                    Invert(
                        name_attribute,
                        std::move(maybe_child_node.value())));
            }
            case DecoratorType::Repeat:
            {
                std::string amount_string = node.attribute("amount").as_string();
                // Convert to lower case: https://stackoverflow.com/a/313990
                std::transform(amount_string.begin(), amount_string.end(), amount_string.begin(),
                    [](unsigned char c)
                    { return std::tolower(c); });
                unsigned long count = 0;
                if (amount_string == "inf")
                {
                    count = std::numeric_limits<unsigned long>::max();
                }
                else
                {
                    try
                    {
                        count = std::stoll(amount_string);
                        if (count < 0)
                        {
                            return tl::make_unexpected(fmt::format(R"(Invalid amount: '{}' | Repeat:['{}',{}])", amount_string, name_attribute, index));
                        }
                    }
                    catch (const std::exception& _)
                    {
                        return tl::make_unexpected(fmt::format(R"(Invalid amount: '{}' | Repeat:['{}',{}])", amount_string, name_attribute, index));
                    }
                }

                return std::make_shared<Repeat>(
                    Repeat(
                        name_attribute,
                        std::move(maybe_child_node.value()),
                        count,
                        node.attribute("break_on_fail").as_bool(false)));
            }
            default:
                return tl::make_unexpected(fmt::format(R"(Got an invalid Decorator node type | {}:['{}',{}])", std::string(node.name()), name_attribute, index));
            }
        }

        tl::expected<std::shared_ptr<node::blackboard::BlackboardNode>, std::string> parseBlackboardNode(const pugi::xml_node& node, const int index, const BlackboardType blackboard_node_type)
        {
            const std::string name_attribute = node.attribute("name").as_string();
```

```cpp
        const std::string variable_name = node.attribute("variable_name").as_string();
        if (variable_name.empty()) {
            return tl::make_unexpected(fmt::format(R"(variable_name cannot be empty | Blackboard:ChangeInteger:[’{}’,{}])", name_attribute, index));
        }
        const int value = node.attribute("value").as_int();
        switch (blackboard_node_type)
        {
        case BlackboardType::ChangeInteger:
        {
            const tl::expected<IntegerChangeType, std::string> maybe_integer_change_type = [&]() {
                const std::string integer_change_type_attribute = node.attribute("integer_change_type").as_string();
                tl::expected<IntegerChangeType, std::string> result;
                switch (utils::hash(integer_change_type_attribute))
                {
                case utils::hash("Set"):
                {
                    result = IntegerChangeType::Set;
                    break;
                }
                case utils::hash("Add"):
                {
                    result = IntegerChangeType::Add;
                    break;
                }
                case utils::hash("Subtract"):
                {
                    result = IntegerChangeType::Subtract;
                    break;
                }
                default:
                {
                    result = tl::make_unexpected(fmt::format(R"(Invalid integer_change_type: ’{}’ | Blackboard:ChangeInteger:[’{}’,{}])", integer_change_type_attribute, name_attribute, index));
                    break;
                }
                };
                return result;
                }
            ();
            if (!maybe_integer_change_type.has_value()) {
                return tl::make_unexpected(maybe_integer_change_type.error());
            }
            return std::make_shared<ChangeInteger>(ChangeInteger(
                name_attribute,
                variable_name,
                maybe_integer_change_type.value(),
                value
            ));
        }
        case BlackboardType::IntegerCondition:
        {
            const tl::expected<ConditionOperatorType, std::string> maybe_condition_operator_type = [&]() {
                const std::string condition_operator_type_attribute = node.attribute("condition_operator_type").as_string();
                tl::expected<ConditionOperatorType, std::string> result;
                switch (utils::hash(condition_operator_type_attribute))
                {
                case utils::hash("="): {
                    result = ConditionOperatorType::Equal;
                    break;
                }
                case utils::hash(">"): {
                    result = ConditionOperatorType::GreaterThan;
                    break;
                }
                case utils::hash(">="): {
                    result = ConditionOperatorType::GreaterThanOrEqual;
                    break;
                }
                case utils::hash("<"): {
                    result = ConditionOperatorType::LessThan;
                    break;
                }
                case utils::hash("<="): {
                    result = ConditionOperatorType::LessThanOrEqual;
                    break;
                }
                case utils::hash("!="): {
                    result = ConditionOperatorType::NotEqual;
                    break;
                }
                default:
                {
                    result = tl::make_unexpected(fmt::format(R"(Invalid condition_operator_type:
```

```
'{}' | Blackboard:IntegerCondition:['{}',{}])", condition_operator_type_attribute, name_attribute,
index));
                            break;
                        }
                        };
                        return result;
                        }
                    ();
                    if (!maybe_condition_operator_type.has_value()) {
                        return tl::make_unexpected(maybe_condition_operator_type.error());
                    }
                    return std::make_shared<IntegerCondition>(IntegerCondition(
                        name_attribute,
                        variable_name,
                        maybe_condition_operator_type.value(),
                        value
                    ));
                }
                default:
                    return tl::make_unexpected(fmt::format(R"(Got an invalid Blackboard node type |
{}:['{}',{}])", std::string(node.name()), name_attribute, index));
                }
        }

        tl::expected<std::shared_ptr<Composite>, std::string> parseComposite(const pugi::xml_node& node,
const int index, const CompositeType composite_type)
        {
            const std::string name_attribute = node.attribute("name").as_string();
            std::vector<std::shared_ptr<node::Node» children;
            for (pugi::xml_node child = node.first_child(); child; child = child.next_sibling())
            {
                auto maybe_node = parseChild(child, STARTING_INDEX + children.size());
                if (!maybe_node.has_value())
                {
                    return tl::make_unexpected(fmt::format(R"({}<-{}:['{}',{}])", maybe_node.error(),
std::string(node.name()), name_attribute, index));
                }
                children.push_back(std::move(maybe_node.value()));
            }
            if (children.size() <= 0)
            {
                return tl::make_unexpected(fmt::format(R"(Composite node must have at least one child |
{}:['{}',{}])", std::string(node.name()), name_attribute, index));
            }
            switch (composite_type)
            {
            case CompositeType::Sequence:
                return std::make_shared<Sequence>(Sequence(name_attribute, std::move(children)));
            case CompositeType::Selector:
                return std::make_shared<Selector>(Selector(name_attribute, std::move(children)));
            case CompositeType::Random:
                return std::make_shared<Random>(Random(name_attribute, std::move(children)));
            default:
                return tl::make_unexpected(fmt::format(R"(Got an invalid Composite node type |
{}:['{}',{}])", std::string(node.name()), name_attribute, index));
            }
        }

        tl::expected<std::shared_ptr<node::custom::CustomNode>, std::string> parseCustomNode(const
pugi::xml_node& node, const int index)
        {
            return this->custom_node_parser->parseCustomNode(node, index);
        }
    };
}

#endif
```

8.5 behaviour_tree/include/behaviour_tree/Context.cpp File Reference

```
#include "Context.h"
#include "BehaviourTree.hpp"
#include "node/Node.hpp"
#include "node/composite/Composite.hpp"
```

Namespaces

- namespace behaviour_tree

8.6 behaviour_tree/include/behaviour_tree/Context.h File Reference

```
#include <algorithm>
#include <memory>
#include <string>
#include <vector>
#include <variant>
#include <unordered_map>
#include "Status.hpp"
```

Classes

- class behaviour_tree::Context

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::composite

8.7 Context.h

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_CONTEXT_HPP
#define BEHAVIOUR_TREE_CONTEXT_HPP

#pragma once

#include <algorithm>
#include <memory>
#include <string>
#include <vector>
#include <variant>
#include <unordered_map>

#include "Status.hpp"

namespace behaviour_tree
{
    class BehaviourTree;
    namespace node
    {
        class Node;
        namespace composite
        {
            class Composite;
        }
    }
}

namespace behaviour_tree
{
    class Context : public std::enable_shared_from_this<Context>
    {
    public:
        Context(std::shared_ptr<BehaviourTree> behaviour_tree);
        void update(const int tick_count);
        Status UseRoot(const int tick_count, const std::string& id);

        std::shared_ptr<BehaviourTree> getBehaviourTree() const { return this->behaviour_tree; };

        void pushNodeTrace(std::pair<std::shared_ptr<node::Node>, int> node_trace);
        void popNode();

        bool canRun() const;

        void setBlackboard(const std::string& key, const int value);
        int getBlackboard(const std::string& key) const;
        int containsBlackboard(const std::string& key) const;

        // Necessary for the class to be a polymorphic
        virtual void _() {};
    private:
        std::shared_ptr<BehaviourTree> behaviour_tree;
        std::vector<std::pair<std::shared_ptr<node::Node>, int» node_trace_list;

        std::unordered_map<std::string, int> blackboard;
    };
}

#endif
```

8.8 behaviour_tree/include/behaviour_tree/node/blackboard/↩ BlackboardNode.hpp File Reference

```
#include "../Node.hpp"
#include "BlackboardType.hpp"
```

Classes

- class behaviour_tree::node::blackboard::BlackboardNode

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::blackboard

8.9 BlackboardNode.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_BLACKBOARD_NODE_HPP
#define BEHAVIOUR_TREE_BLACKBOARD_NODE_HPP

#pragma once

#include "../Node.hpp"

#include "BlackboardType.hpp"

namespace behaviour_tree::node::blackboard
{
    class BlackboardNode : public Node
    {
    public:
        BlackboardNode(const std::string& name) : Node(name)
        {
        }

        virtual const BlackboardType type() const = 0;
    };
}

#endif
```

8.10 behaviour_tree/include/behaviour_tree/node/blackboard/←BlackboardType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::blackboard

Enumerations

- enum class behaviour_tree::node::blackboard::BlackboardType { behaviour_tree::node::blackboard::ChangeInteger , behaviour_tree::node::blackboard::IntegerCondition }

8.11 BlackboardType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_BLACKBOARD_TYPE_HPP
#define BEHAVIOUR_TREE_BLACKBOARD_TYPE_HPP

#pragma once

namespace behaviour_tree::node::blackboard
{
    enum class BlackboardType
    {
        ChangeInteger,
        IntegerCondition,
     };
}

#endif
```

8.12 behaviour_tree/include/behaviour_tree/node/blackboard/Change←Integer.hpp File Reference

```
#include "BlackboardNode.hpp"
#include "enum/IntegerChangeType.hpp"
```

Classes

- class behaviour_tree::node::blackboard::ChangeInteger

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::blackboard

8.13 ChangeInteger.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_CHANGE_INTEGER_HPP
#define BEHAVIOUR_TREE_CHANGE_INTEGER_HPP

#pragma once

#include "BlackboardNode.hpp"

#include "enum/IntegerChangeType.hpp"

namespace behaviour_tree::node::blackboard
{
    class ChangeInteger final : public BlackboardNode
    {
    public:
        ChangeInteger(const std::string& name, const std::string variable_name, const IntegerChangeType integer_change_type, const int value) : BlackboardNode(name), variable_name(variable_name), integer_change_type(integer_change_type), value(value) {}

        const BlackboardType type() const final override { return BlackboardType::ChangeInteger; }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            if (!context->containsBlackboard(this->variable_name)) {
                context->setBlackboard(this->variable_name, 0);
            }
            int value = context->getBlackboard(this->variable_name);
            switch (this->integer_change_type) {
            case IntegerChangeType::Set: {
                context->setBlackboard(this->variable_name, this->value);
                break;
            }
            case IntegerChangeType::Add: {
                context->setBlackboard(this->variable_name, value + this->value);
                break;
            }
            case IntegerChangeType::Subtract: {
                context->setBlackboard(this->variable_name, value - this->value);
                break;
            }
            }
            return Status::Success;
        }

        const std::string toString() const final override
        {
            const std::string& name = this->getName();
            const std::string integer_change_type = [&]() {
                switch (this->integer_change_type)
                {
                case IntegerChangeType::Set:
                    return "Set";
                case IntegerChangeType::Add:
                    return "Add";
                case IntegerChangeType::Subtract:
                    return "Subtract";
                default:
                    return "Invalid";
                }
                }
            ();
            if (name != "")
                return fmt::format(R"(<Blackboard:ChangeInteger name='{}' variable_name='{}' integer_change_type='{}' value='{}'/>)", name, this->variable_name, integer_change_type, this->value);
            else
                return fmt::format(R"(<Blackboard:ChangeInteger variable_name='{}' integer_change_type='{}' value='{}'/>)", this->variable_name, integer_change_type, this->value);
        }

    private:
        const std::string variable_name;
        const IntegerChangeType integer_change_type;
        const int value;
    };
}

#endif
```

8.14 behaviour_tree/include/behaviour_tree/node/blackboard/enum/↩ ConditionOperatorType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::blackboard

Enumerations

- enum class behaviour_tree::node::blackboard::ConditionOperatorType {
  behaviour_tree::node::blackboard::Equal , behaviour_tree::node::blackboard::NotEqual , behaviour_tree::node::blackboard::Gre
  , behaviour_tree::node::blackboard::GreaterThanOrEqual ,
  behaviour_tree::node::blackboard::LessThan , behaviour_tree::node::blackboard::LessThanOrEqual }

8.15 ConditionOperatorType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_CONDITION_OPERATOR_TYPE_HPP
#define BEHAVIOUR_TREE_CONDITION_OPERATOR_TYPE_HPP

#pragma once

namespace behaviour_tree::node::blackboard
{
    enum class ConditionOperatorType
    {
        Equal,
        NotEqual,
        GreaterThan,
        GreaterThanOrEqual,
        LessThan,
        LessThanOrEqual,
     };
 }

#endif
```

8.16 behaviour_tree/include/behaviour_tree/node/blackboard/enum/↩ IntegerChangeType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::blackboard

Enumerations

- enum class behaviour_tree::node::blackboard::IntegerChangeType { behaviour_tree::node::blackboard::Set ,
  behaviour_tree::node::blackboard::Add , behaviour_tree::node::blackboard::Subtract }

8.17 IntegerChangeType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_INTEGER_CHANGE_TYPE_HPP
#define BEHAVIOUR_TREE_INTEGER_CHANGE_TYPE_HPP

#pragma once

namespace behaviour_tree::node::blackboard
{
    enum class IntegerChangeType
    {
        Set,
        Add,
        Subtract,
    };
}

#endif
```

8.18 behaviour_tree/include/behaviour_tree/node/blackboard/Integer←Condition.hpp File Reference

```
#include "BlackboardNode.hpp"
#include "enum/ConditionOperatorType.hpp"
```

Classes

- class behaviour_tree::node::blackboard::IntegerCondition

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::blackboard

8.19 IntegerCondition.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_INTEGER_CONDITION_HPP
#define BEHAVIOUR_TREE_INTEGER_CONDITION_HPP

#pragma once

#include "BlackboardNode.hpp"

#include "enum/ConditionOperatorType.hpp"

namespace behaviour_tree::node::blackboard
{
    class IntegerCondition final : public BlackboardNode
    {
    public:
        IntegerCondition(const std::string& name, const std::string variable_name, const
    ConditionOperatorType condition_operator, const int value) : BlackboardNode(name),
    variable_name(variable_name), condition_operator(condition_operator), value(value) {}

        const BlackboardType type() const final override { return BlackboardType::IntegerCondition; }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            if (!context->containsBlackboard(this->variable_name)) {
                context->setBlackboard(this->variable_name, 0);
            }
            switch (this->condition_operator)
            {
            case ConditionOperatorType::Equal:
                return context->getBlackboard(this->variable_name) == this->value ? Status::Success :
       Status::Failure;
            case ConditionOperatorType::GreaterThan:
                return context->getBlackboard(this->variable_name) > this->value ? Status::Success :
       Status::Failure;
            case ConditionOperatorType::GreaterThanOrEqual:
                return context->getBlackboard(this->variable_name) >= this->value ? Status::Success :
       Status::Failure;
            case ConditionOperatorType::LessThan:
                return context->getBlackboard(this->variable_name) < this->value ? Status::Success :
       Status::Failure;
            case ConditionOperatorType::LessThanOrEqual:
                return context->getBlackboard(this->variable_name) <= this->value ? Status::Success :
       Status::Failure;
            case ConditionOperatorType::NotEqual:
                return context->getBlackboard(this->variable_name) != this->value ? Status::Success :
       Status::Failure;
            default:
                return Status::Failure;
            }
        }

        const std::string toString() const final override
        {
            const std::string& name = this->getName();
            const std::string condition_operator = [&]() {
                switch (this->condition_operator)
                {
                case ConditionOperatorType::Equal:
                    return "=";
                case ConditionOperatorType::GreaterThan:
                    return ">";
                case ConditionOperatorType::GreaterThanOrEqual:
                    return ">=";
                case ConditionOperatorType::LessThan:
                    return "<";
                case ConditionOperatorType::LessThanOrEqual:
                    return "<=";
                case ConditionOperatorType::NotEqual:
                    return "!=";
                default:
                    return "Invalid";
                }
                }
            ();
            if (name != "")
                return fmt::format(R"(<Blackboard:IntegerCondition name='{}' variable_name='{}'
       condition_operator='{}' value='{}'/>)", name, this->variable_name, condition_operator, this->value);
            else
                return fmt::format(R"(<Blackboard:IntegerCondition variable_name='{}'
       condition_operator='{}' value='{}'/>)", this->variable_name, condition_operator, this->value);
        }

    private:
        const std::string variable_name;
        const ConditionOperatorType condition_operator;
        const int value;
    };
}

#endif
```

8.20 behaviour_tree/include/behaviour_tree/node/composite/↵ Composite.hpp File Reference

```
#include <vector>
#include "../Node.hpp"
#include "CompositeType.hpp"
```

Classes

- class behaviour_tree::node::composite::Composite

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::composite

8.21 Composite.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_COMPOSITE_HPP
#define BEHAVIOUR_TREE_COMPOSITE_HPP

#pragma once

#include <vector>

#include "../Node.hpp"
#include "CompositeType.hpp"

namespace behaviour_tree::node::composite
{
    class Composite : public Node
    {
    public:
        Composite(const std::string& name, std::vector<std::shared_ptr<Node» children) : Node(name),
    children(std::move(children))
        {
        }

        Status tick(const int tick_count, std::shared_ptr<Context> context, const int start_index)
        {
            assert(context != nullptr);

            if (!this->started) {
                this->start(context);
                this->started = true;
            }
            Status status = this->run(tick_count, context, start_index);
            if (status == Status::Success || status == Status::Failure) {
                this->finish(context);
                this->started = false;
            }
            return status;
        }

        virtual const Status run(const int tick_count, std::shared_ptr<Context> context, const int
    start_index) = 0;
        virtual const CompositeType type() const = 0;

    protected:
        std::vector<std::shared_ptr<Node» children;
    };
}

#endif
```

8.22 behaviour_tree/include/behaviour_tree/node/composite/↩ CompositeType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::composite

Enumerations

- enum class behaviour_tree::node::composite::CompositeType { behaviour_tree::node::composite::Sequence , behaviour_tree::node::composite::Selector , behaviour_tree::node::composite::Random }

8.23 CompositeType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_COMPOSITETYPE_HPP
#define BEHAVIOUR_TREE_COMPOSITETYPE_HPP

#pragma once

namespace behaviour_tree::node::composite
{
    enum class CompositeType
    {
        Sequence,
        Selector,
        Random,
     };
 }

 #endif
```

8.24 behaviour_tree/include/behaviour_tree/node/composite/↩ Random.hpp File Reference

```
#include <effolkronium/random.hpp>
#include "Composite.hpp"
```

Classes

- class behaviour_tree::node::composite::Random

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::composite

Typedefs

- using effolkronium_Random = effolkronium::random_static

8.24.1 Typedef Documentation

8.24.1.1 effolkronium_Random

```
using effolkronium_Random = effolkronium::random_static
```

8.25 Random.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_RANDOM_HPP
#define BEHAVIOUR_TREE_RANDOM_HPP

#pragma once

#include <effolkronium/random.hpp>

#include "Composite.hpp"

using effolkronium_Random = effolkronium::random_static;

namespace behaviour_tree::node::composite
{
    class Random final : public Composite
    {
    public:
        Random(const std::string& name, std::vector<std::shared_ptr<Node>> children) : Composite(name,
    std::move(children)) {}

        const CompositeType type() const final override { return CompositeType::Random; }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            return this->run(tick_count, context, 0);
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context, const int start_index)
    final override
        {
            // Random can never be empty from the BehaviourTreeParser
            return this->children[effolkronium_Random::get(0, static_cast<int>(this->children.size() -
    1))]->tick(tick_count, context);
        }

        const std::string toString() const final override {
            std::string out;
            for (auto& child : this->children)
            {
                out += child->toString();
            }
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<Random name='{}'>{}</Random>)", name, out);
            else
                return fmt::format(R"(<Random>{}</Random>)", out);
        }
    };
}

#endif
```

8.26 behaviour_tree/include/behaviour_tree/node/composite/↩ Selector.hpp File Reference

```
#include "Composite.hpp"
```

Classes

- class behaviour_tree::node::composite::Selector

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::composite

8.27 Selector.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SELECTOR_HPP
#define BEHAVIOUR_TREE_SELECTOR_HPP

#pragma once

#include "Composite.hpp"

namespace behaviour_tree::node::composite
{
    class Selector final : public Composite
    {
    public:
        Selector(const std::string& name, std::vector<std::shared_ptr<Node>> children) : Composite(name,
    std::move(children)) {}

        const CompositeType type() const final override { return CompositeType::Selector; }

        void start(std::shared_ptr<Context> context) final override {
            this->previous_start_index = -1;
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            return this->run(tick_count, context, 0);
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context, const int start_index)
    final override
        {
            for (int i = start_index; i < this->children.size(); i++)
            {
                if (this->previous_start_index != i) {
                    context->pushNodeTrace(std::make_pair(shared_from_this(), i));
                }
                auto& child = this->children[i];
                auto status = child->tick(tick_count, context);
                switch (status) {
                case Status::Running:
                    this->previous_start_index = i;
                    return Status::Running;
                case Status::Failure:
                    context->popNode();
                    continue;
                case Status::Success:
                    context->popNode();
                    return Status::Success;
                }
            }
            return Status::Failure;
        }

        const std::string toString() const final override {
            std::string out;
            for (auto& child : this->children)
            {
                out += child->toString();
            }
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<Selector name='{}'>{}</Selector>)", name, out);
            else
                return fmt::format(R"(<Selector>{}</Selector>)", out);
        }

    private:
        int previous_start_index = -1;
    };
}

#endif
```

8.28 behaviour_tree/include/behaviour_tree/node/composite/↩ Sequence.hpp File Reference

```
#include "Composite.hpp"
```

Classes

- class behaviour_tree::node::composite::Sequence

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::composite

8.29 Sequence.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SEQUENCE_HPP
#define BEHAVIOUR_TREE_SEQUENCE_HPP

#pragma once

#include "Composite.hpp"

namespace behaviour_tree::node::composite
{
    class Sequence final : public Composite
    {
    public:
        Sequence(const std::string& name, std::vector<std::shared_ptr<Node>> children) : Composite(name,
    std::move(children)) {}

        const CompositeType type() const final override { return CompositeType::Sequence; }

        void start(std::shared_ptr<Context> context) final override {
            this->previous_start_index = -1;
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            return this->run(tick_count, context, 0);
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context, const int start_index)
    final override
        {
            for (int i = start_index; i < this->children.size(); i++)
            {
                if (this->previous_start_index != i) {
                    context->pushNodeTrace(std::make_pair(shared_from_this(), i));
                }
                auto& child = this->children[i];
                auto status = child->tick(tick_count, context);
                switch (status) {
                case Status::Running:
                    this->previous_start_index = i;
                    return Status::Running;
                case Status::Success:
                    context->popNode();
                    continue;
                case Status::Failure:
                    context->popNode();
                    return Status::Failure;
                }
            }
            return Status::Success;
        }

        const std::string toString() const final override {
            std::string out;
            for (auto& child : this->children)
            {
                out += child->toString();
            }
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<Sequence name='{}'>{}</Sequence>)", name, out);
            else
                return fmt::format(R"(<Sequence>{}</Sequence>)", out);
        }

    private:
        int previous_start_index = -1;
    };
}

#endif
```

8.30 behaviour_tree/include/behaviour_tree/node/custom/Custom↩ Node.hpp File Reference

```
#include "../Node.hpp"
```

Classes

- class behaviour_tree::node::custom::CustomNode

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom

8.31 CustomNode.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_CUSTOM_NODE_HPP
#define BEHAVIOUR_TREE_CUSTOM_NODE_HPP

#pragma once

#include "../Node.hpp"

namespace behaviour_tree::node::custom
{
    class CustomNode : public Node
    {
    public:
        CustomNode(const std::string& name) : Node(name) {

        }
    };
}

#endif
```

8.32 behaviour_tree/include/behaviour_tree/node/custom/CustomNode↩ Parser.hpp File Reference

```
#include <string>
#include <memory>
#include <pugixml.hpp>
#include <tl/expected.hpp>
#include <fmt/format.h>
#include "CustomNode.hpp"
```

Classes

- class behaviour_tree::node::custom::CustomNodeParser

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom

8.33 CustomNodeParser.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_CUSTOM_NODE_PARSER_HPP
#define BEHAVIOUR_TREE_CUSTOM_NODE_PARSER_HPP

#pragma once

#include <string>
#include <memory>

#include <pugixml.hpp>
#include <tl/expected.hpp>
#include <fmt/format.h>

#include "CustomNode.hpp"

namespace behaviour_tree::node::custom
{
    class CustomNodeParser
    {
    public:
        virtual tl::expected<std::shared_ptr<CustomNode>, std::string> parseCustomNode(const
    pugi::xml_node& node, const int index) = 0;
    };
}

#endif
```

8.34 behaviour_tree/include/behaviour_tree/node/decorator/↩ Decorator.hpp File Reference

```
#include "../Node.hpp"
#include "DecoratorType.hpp"
```

Classes

- class behaviour_tree::node::decorator::Decorator

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::decorator

8.35 Decorator.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_DECORATOR_HPP
#define BEHAVIOUR_TREE_DECORATOR_HPP

#pragma once

#include "../Node.hpp"

#include "DecoratorType.hpp"

namespace behaviour_tree::node::decorator
{
    class Decorator : public Node
    {
    public:
        Decorator(const std::string& name, std::shared_ptr<Node> child) : Node(name),
    child(std::move(child))
        {
        }

        virtual const DecoratorType type() const = 0;

    protected:
        std::shared_ptr<Node> child;
    };
}

#endif
```

8.36 behaviour_tree/include/behaviour_tree/node/decorator/Decorator↩ Type.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::decorator

Enumerations

- enum class behaviour_tree::node::decorator::DecoratorType { behaviour_tree::node::decorator::Invert , behaviour_tree::node::decorator::Repeat }

8.37 DecoratorType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_DECORATORTYPE_HPP
#define BEHAVIOUR_TREE_DECORATORTYPE_HPP

#pragma once

namespace behaviour_tree::node::decorator
{
    enum class DecoratorType
    {
        Invert,
        Repeat,
     };
}

#endif
```

8.38 behaviour_tree/include/behaviour_tree/node/decorator/Invert.hpp File Reference

```
#include "Decorator.hpp"
```

Classes

- class behaviour_tree::node::decorator::Invert

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::decorator

8.39 Invert.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_INVERT_HPP
#define BEHAVIOUR_TREE_INVERT_HPP

#pragma once

#include "Decorator.hpp"

namespace behaviour_tree::node::decorator
{
     class Invert final : public Decorator
     {
     public:
         Invert(const std::string& name, std::shared_ptr<Node> child) : Decorator(name, std::move(child))
     {}

         const DecoratorType type() const final override { return DecoratorType::Invert; }

         void start(std::shared_ptr<Context> context) final override {
             context->pushNodeTrace(std::make_pair(shared_from_this(), 0));
         }

         void finish(std::shared_ptr<Context> context) final override {
             context->popNode();
         }
```

```

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            auto status = this->child->tick(tick_count, context);
            switch (status)
            {
            case Status::Success:
                return Status::Failure;
            case Status::Failure:
                return Status::Success;
            case Status::Running:
                return Status::Running;
            }
        }

        const std::string toString() const final override {
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<Invert name='{}'>{}</Invert>)", name, this->child->toString());
            else
                return fmt::format(R"(<Invert>{}</Invert>)", this->child->toString());
        }
    };
}

#endif
```

8.40 behaviour_tree/include/behaviour_tree/node/decorator/Repeat.hpp File Reference

```
#include <numeric>
#include "Decorator.hpp"
```

Classes

- class behaviour_tree::node::decorator::Repeat

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::decorator

8.41 Repeat.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_REPEAT_HPP
#define BEHAVIOUR_TREE_REPEAT_HPP

#pragma once

#include <numeric>

#include "Decorator.hpp"

namespace behaviour_tree::node::decorator
{
    class Repeat final : public Decorator
    {
    public:
        Repeat(const std::string& name, std::shared_ptr<Node> child, const unsigned long amount, const
    bool break_on_fail) : Decorator(name, std::move(child)), amount(amount), break_on_fail(break_on_fail)
        {
        }

        const DecoratorType type() const final override { return DecoratorType::Repeat; }

        void start(std::shared_ptr<Context> context) final override {
            this->count = 0;
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            auto status = this->child->tick(tick_count, context);
            switch (status)
            {
            case Status::Success:
                this->count++;
                if (this->count >= this->amount)
                {
                    return Status::Success;
                }
                return Status::Running;
            case Status::Running:
                return Status::Running;
            case Status::Failure:
                if (this->break_on_fail)
                    return Status::Failure;
                return Status::Running;
            }
        }

        const std::string toString() const final override {
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<Repeat name='{}' amount='{}' break_on_fail='{}'>{}</Repeat>)", name, this->getAmount(), this->getBreakOnFail() ? "true" : "false", this->child->toString());
            else
                return fmt::format(R"(<Repeat amount='{}' break_on_fail='{}'>{}</Repeat>)", this->getAmount(), this->getBreakOnFail() ? "true" : "false", this->child->toString());
        }

        const unsigned long getAmount() const {
            return this->amount;
        }

        const bool getBreakOnFail() const {
            return this->break_on_fail;
        }

        const unsigned long getCount() const {
            return this->count;
        }

    private:
        const unsigned long amount;
        const bool break_on_fail;

        unsigned long count = 0;
    };
}

#endif
```

8.42 behaviour_tree/include/behaviour_tree/node/leaf/Fail.hpp File Reference

```
#include "Leaf.hpp"
```

Classes

- class behaviour_tree::node::leaf::Fail

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::leaf

8.43 Fail.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_FAIL_HPP
#define BEHAVIOUR_TREE_FAIL_HPP

#pragma once

#include "Leaf.hpp"

namespace behaviour_tree::node::leaf
{
    class Fail final : public Leaf
    {
    public:
        Fail(const std::string& name) : Leaf(name) {}

        const LeafType type() const final override { return LeafType::Fail; }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            return Status::Failure;
        }

        const std::string toString() const final override {
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<Fail name='{}'/>)", name);
            else
                return fmt::format(R"(<Fail/>)");
        }
    };
}

#endif
```

8.44 behaviour_tree/include/behaviour_tree/node/leaf/Leaf.hpp File Reference

```
#include "../Node.hpp"
#include "LeafType.hpp"
```

Classes

- class behaviour_tree::node::leaf::Leaf

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::leaf

8.45 Leaf.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_LEAF_HPP
#define BEHAVIOUR_TREE_LEAF_HPP

#pragma once

#include "../Node.hpp"

#include "LeafType.hpp"

namespace behaviour_tree::node::leaf
{
    class Leaf : public Node
    {
    public:
        Leaf(const std::string& name) : Node(name)
        {
        }

        virtual const LeafType type() const = 0;
    };
}

#endif
```

8.46 behaviour_tree/include/behaviour_tree/node/leaf/LeafType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::leaf

Enumerations

- enum class behaviour_tree::node::leaf::LeafType { behaviour_tree::node::leaf::Succeed , behaviour_tree::node::leaf::Fail , behaviour_tree::node::leaf::UseRoot }

8.47 LeafType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_LEAFTYPE_HPP
#define BEHAVIOUR_TREE_LEAFTYPE_HPP

#pragma once

namespace behaviour_tree::node::leaf
{
    enum class LeafType
    {
        Succeed,
        Fail,
        UseRoot,
    };
}

#endif
```

8.48 behaviour_tree/include/behaviour_tree/node/leaf/Succeed.hpp File Reference

```
#include "Leaf.hpp"
```

Classes

- class behaviour_tree::node::leaf::Succeed

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::leaf

8.49 Succeed.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SUCCEED_HPP
#define BEHAVIOUR_TREE_SUCCEED_HPP

#pragma once

#include "Leaf.hpp"

namespace behaviour_tree::node::leaf
{
    class Succeed final : public Leaf
    {
    public:
        Succeed(const std::string& name) : Leaf(name) {}

        const LeafType type() const final override { return LeafType::Succeed; }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            return Status::Success;
        }

        const std::string toString() const final override {
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<Succeed name='{}'/>)", name);
            else
                return fmt::format(R"(<Succeed/>)");
        }
    };
}

#endif
```

8.50 behaviour_tree/include/behaviour_tree/node/leaf/UseRoot.hpp File Reference

```
#include "Leaf.hpp"
```

Classes

- class behaviour_tree::node::leaf::UseRoot

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::leaf

8.51 UseRoot.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_UseRoot_HPP
#define BEHAVIOUR_TREE_UseRoot_HPP

#pragma once

#include "Leaf.hpp"

namespace behaviour_tree::node::leaf
{
    class UseRoot final : public Leaf
    {
    public:
        UseRoot(const std::string& name, const std::string& id) : Leaf(name), id(id) {}

        const LeafType type() const final override { return LeafType::UseRoot; }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            return context->UseRoot(tick_count, this->id);
        }

        const std::string& getId() const { return this->id; }

        const std::string toString() const final override {
            const std::string& name = this->getName();
            if (name != "")
                return fmt::format(R"(<UseRoot name='{}' id='{}'/>)", name, this->getId());
            else
                return fmt::format(R"(<UseRoot id='{}'/>)", this->getId());
        }

    private:
        const std::string id;
    };
}

#endif
```

8.52 behaviour_tree/include/behaviour_tree/node/Node.hpp File Reference

```
#include <cassert>
#include <string>
#include <memory>
#include <fmt/format.h>
#include "../Status.hpp"
#include "../Context.h"
```

Classes

- class behaviour_tree::node::Node

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node

8.53 Node.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_NODE_HPP
#define BEHAVIOUR_TREE_NODE_HPP

#pragma once

#include <cassert>

#include <string>
#include <memory>

#include <fmt/format.h>

#include "../Status.hpp"
#include "../Context.h"

namespace behaviour_tree::node
{
    class Node : public std::enable_shared_from_this<Node>
    {
    public:
        Node(const std::string& name) : name(name)
        {
        }

        virtual void start(std::shared_ptr<Context> context) {}
        virtual const Status run(const int tick_count, std::shared_ptr<Context> context) = 0;
        virtual void finish(std::shared_ptr<Context> context) {}

        Status tick(const int tick_count, std::shared_ptr<Context> context)
        {
            assert(context != nullptr);

            if (!this->started) {
                this->start(context);
                this->started = true;
            }
            Status status = this->run(tick_count, context);
            if (status == Status::Success || status == Status::Failure) {
                this->finish(context);
                this->started = false;
            }
            return status;
        }

        virtual const std::string toString() const = 0;

        const std::string& getName() const
        {
            return this->name;
        }

    protected:
        const std::string name;

        bool started = false;
    };
}

#endif
```

8.54 behaviour_tree/include/behaviour_tree/Root.hpp File Reference

```
#include <string>
#include <fmt/format.h>
#include "node/Node.hpp"
```

Classes

- class behaviour_tree::Root

Namespaces

- namespace behaviour_tree

8.55 Root.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_ROOT_HPP
#define BEHAVIOUR_TREE_ROOT_HPP

#pragma once

#include <string>
#include <fmt/format.h>

#include "node/Node.hpp"

using namespace behaviour_tree::node;

namespace behaviour_tree
{
    class Root
    {
    public:
        Root(std::string id, std::shared_ptr<node::Node> child) : id(id), child(std::move(child))
        {
        }

        const Status tick(const int tick_count, std::shared_ptr<Context>& context)
        {
            if (this->child == nullptr) return Status::Success;
            return this->child->tick(tick_count, context);
        }

        const std::string toString() const {
            const std::string& id = this->getId();
            std::string child_string = this->child != nullptr ? this->child->toString() : "";
            if (id != "")
                return fmt::format(R"(<Root id='{}'>{}</Root>)", id, child_string);
            else
                return fmt::format(R"(<Root>{}</Root>)", child_string);
        }

        const std::string& getId() const
        {
            return this->id;
        }

    private:
        std::string id;
        std::shared_ptr<node::Node> child;
    };
}

#endif
```

8.56 behaviour_tree/include/behaviour_tree/Status.hpp File Reference

Namespaces

- namespace behaviour_tree

Enumerations

- enum class behaviour_tree::Status { behaviour_tree::Success , behaviour_tree::Failure , behaviour_tree::Running }

8.57 Status.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_STATUS_HPP
#define BEHAVIOUR_TREE_STATUS_HPP

#pragma once

namespace behaviour_tree
{
    enum class Status
    {
        Success,
        Failure,
        Running
     };
}

#endif
```

8.58 behaviour_tree_car/include/behaviour_tree/global/distance.h File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::global

Variables

- constexpr double behaviour_tree::global::CM_TO_DISTANCE = 15.15151515151515151515151515151515151515151515151

8.59 distance.h

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_GLOBAL_DISTANCE_H
#define BEHAVIOUR_TREE_GLOBAL_DISTANCE_H

#pragma once

namespace behaviour_tree::global {
    constexpr double CM_TO_DISTANCE = 15.151515151515151515151515151515151515151515151515151515;
}

#endif
```

8.60 behaviour_tree_car/include/behaviour_tree/node/custom/action/↩ Drive.hpp File Reference

```
#include <fmt/format.h>
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include "enum/DirectionType.hpp"
```

Classes

- class behaviour_tree::node::custom::action::Drive

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

8.61 Drive.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_DRIVE_HPP
#define BEHAVIOUR_TREE_DRIVE_HPP

#pragma once

#include <fmt/format.h>

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

#include "enum/DirectionType.hpp"

namespace behaviour_tree::node::custom::action
{
    class Drive final : public CustomNode
    {
    public:
        Drive(const std::string &name, const int speed, const DirectionType direction_type) :
    CustomNode(name), speed(speed), direction_type(direction_type)
        {
        }

        const static tl::expected<std::shared_ptr<Drive>, std::string> parse(const pugi::xml_node &node,
    const int index, const std::string &name_attribute)
        {
            const int speed = node.attribute("speed").as_int();
            if (speed < 0 || speed > 100)
            {
                return tl::unexpected(fmt::format(R"(Invalid speed: '{}' | Action:Drive:['{}',{}])",
    speed, name_attribute, index));
            }
            const tl::expected<DirectionType, std::string> maybe_direction_type =
                [&]()
            {
                const std::string direction_type_attribute =
    node.attribute("direction_type").as_string();
                tl::expected<DirectionType, std::string> result;
                switch (utils::hash(direction_type_attribute))
                {
                case utils::hash("Forward"):
                {
                    result = DirectionType::Forward;
                    break;
                }
                case utils::hash("Backward"):
                {
                    result = DirectionType::Backward;
                    break;
                }
                default:
                {
                    result = tl::make_unexpected(fmt::format(R"(Invalid direction_type: '{}' | Action:Drive:['{}',{}])", direction_type_attribute, name_attribute, index));
                    break;
                }
                };
                return result;
            }();
            if (!maybe_direction_type.has_value())
            {
                return tl::make_unexpected(maybe_direction_type.error());
            }
            return std::make_shared<Drive>(
                Drive(
                    name_attribute,
                    speed,
                    maybe_direction_type.value()));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();
            const MovementSystem *movement_system = car_system->getMovementSystem();
            switch (this->getDirectionType())
            {
            case DirectionType::Backward:
            {
                movement_system->setRearWheelsDirectionToBackward();
                break;
            }
            case DirectionType::Forward:
            {
                movement_system->setRearWheelsDirectionToForward();
                break;
            }
            }
            movement_system->setRearWheelsSpeed(this->getSpeed());
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Success;
        }

        const std::string toString() const final override
        {
            const std::string &name = this->getName();
            std::string direction_type;
            switch (this->getDirectionType())
            {
            case DirectionType::Forward:
                direction_type = "Forward";
                break;
            case DirectionType::Backward:
                direction_type = "Backward";
                break;
            }
            if (name != "")
                return fmt::format(R"(<Action:Drive name='{}' speed='{}' direction_type='{}'/>)", name, this->getSpeed(), direction_type);
            else
                return fmt::format(R"(<Action:Drive speed='{}' direction_type='{}'/>)", this->getSpeed(), direction_type);
        }

        const int getSpeed() const
        {
            return this->speed;
        }

        const DirectionType getDirectionType() const
        {
            return this->direction_type;
        }

```

```
    private:
        const int speed;
        const DirectionType direction_type;
    };
}

#endif
```

8.62 behaviour_tree_car/include/behaviour_tree/node/custom/action/enum/ClockDirectionType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

Enumerations

- enum class behaviour_tree::node::custom::action::ClockDirectionType { behaviour_tree::node::custom::action::Clockwise , behaviour_tree::node::custom::action::AntiClockwise }

8.63 ClockDirectionType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_CLOCKDIRECTIONTYPE_HPP
#define BEHAVIOUR_TREE_CLOCKDIRECTIONTYPE_HPP

#pragma once

namespace behaviour_tree::node::custom::action
{
    enum class ClockDirectionType
    {
        Clockwise,
        AntiClockwise,
    };
}

#endif
```

8.64 behaviour_tree_car/include/behaviour_tree/node/custom/action/enum/DirectionType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

Enumerations

- enum class behaviour_tree::node::custom::action::DirectionType { behaviour_tree::node::custom::action::Forward , behaviour_tree::node::custom::action::Backward }

8.65 DirectionType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_DIRECTIONTYPE_HPP
#define BEHAVIOUR_TREE_DIRECTIONTYPE_HPP

#pragma once

namespace behaviour_tree::node::custom::action
{
    enum class DirectionType
    {
        Forward,
        Backward,
     };
 }

 #endif
```

8.66 behaviour_tree_car/include/behaviour_tree/node/custom/action/enum/ServoType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

Enumerations

- enum class behaviour_tree::node::custom::action::ServoType { behaviour_tree::node::custom::action::FrontWheels , behaviour_tree::node::custom::action::CameraServo1 , behaviour_tree::node::custom::action::CameraServo2 }

8.67 ServoType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SERVOTYPE_HPP
#define BEHAVIOUR_TREE_SERVOTYPE_HPP

#pragma once

namespace behaviour_tree::node::custom::action
{
    enum class ServoType
    {
        FrontWheels,
        CameraServo1,
        CameraServo2,
     };
 }

 #endif
```

8.68 behaviour_tree_car/include/behaviour_tree/node/custom/action/enum/WheelType.hpp File Reference

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

Enumerations

- enum class behaviour_tree::node::custom::action::WheelType { behaviour_tree::node::custom::action::Left , behaviour_tree::node::custom::action::Right , behaviour_tree::node::custom::action::Both }

8.69 WheelType.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_WHEELTYPE_HPP
#define BEHAVIOUR_TREE_WHEELTYPE_HPP

#pragma once

namespace behaviour_tree::node::custom::action
{
    enum class WheelType
    {
        Left,
        Right,
        Both,
     };
}

#endif
```

8.70 behaviour_tree_car/include/behaviour_tree/node/custom/action/PauseExecution.hpp File Reference

```
#include <chrono>
#include <fmt/format.h>
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
```

Classes

- class behaviour_tree::node::custom::action::PauseExecution

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

8.71 PauseExecution.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_PAUSE_EXECUTION_HPP
#define BEHAVIOUR_TREE_PAUSE_EXECUTION_HPP

#pragma once

#include <chrono>
#include <fmt/format.h>

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"

namespace behaviour_tree::node::custom::action
{
    class PauseExecution final : public CustomNode
    {
    public:
        PauseExecution(const std::string &name, const int ms) : CustomNode(name), ms(ms)
        {
        }

        const static tl::expected<std::shared_ptr<PauseExecution>, std::string> parse(const
    pugi::xml_node &node, const int index, const std::string &name_attribute)
        {
            int ms = node.attribute("ms").as_int();
            if (ms < 0)
            {
                return tl::unexpected(fmt::format(R"(Invalid ms: '{}' |
    Action:PauseExecution:['{}',{}])", ms, name_attribute, index));
            }
            return std::make_shared<custom::action::PauseExecution>(
                custom::action::PauseExecution(
                    name_attribute,
                    ms));
        }

        void start(std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            this->start_time = std::chrono::steady_clock::now();
#endif
        }

        void finish(std::shared_ptr<Context> context) final override
        {
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            if (std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::steady_clock::now() -
    this->start_time).count() < this->ms)
            {
                return Status::Running;
            }
#endif
            return Status::Success;
        }

        const std::string toString() const final override
        {
            const std::string &name = this->getName();
            if (name != "")
                return fmt::format(R"(<Action:PauseExecution name='{}' ms='{}'/>)", name, this->getMS());
            else
                return fmt::format(R"(<Action:PauseExecution ms='{}'/>)", this->getMS());
        }

        const int getMS() const
        {
            return this->ms;
        }

    private:
        std::chrono::steady_clock::time_point start_time;
        const int ms;
    };
}

#endif
```

8.72 behaviour_tree_car/include/behaviour_tree/node/custom/action/↩ Print.hpp File Reference

```
#include <iostream>
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
```

Classes

- class behaviour_tree::node::custom::action::Print

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

8.73 Print.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_PRINT_HPP
#define BEHAVIOUR_TREE_PRINT_HPP

#pragma once

#include <iostream>

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

namespace behaviour_tree::node::custom::action
{
    class Print final : public CustomNode
    {
    public:
        Print(const std::string &name, const std::string &text) : CustomNode(name), text(text)
        {
        }

        const static tl::expected<std::shared_ptr<Print>, std::string> parse(const pugi::xml_node &node,
    const int index, const std::string &name_attribute)
        {
            return std::make_shared<Print>(
                Print(
                    name_attribute,
                    node.attribute("text").as_string()));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            spdlog::info(this->getText());
#else
            std::cout « this->getText();
#endif
```

```
            return Status::Success;
        }

        const std::string &getText() const
        {
            return this->text;
        }

        const std::string toString() const final override
        {
            const std::string &name = this->getName();
            if (name != "")
                return fmt::format(R"(<Action:Print name='{}' text='{}'/>)", name, this->getText());
            else
                return fmt::format(R"(<Action:Print text='{}'/>)", this->getText());
        }

    private:
        const std::string text;
    };
}

#endif
```

8.74 behaviour_tree_car/include/behaviour_tree/node/custom/action/↩ SetAngle.hpp File Reference

```
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include "enum/ServoType.hpp"
```

Classes

- class behaviour_tree::node::custom::action::SetAngle

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

8.75 SetAngle.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SETANGLE_HPP
#define BEHAVIOUR_TREE_SETANGLE_HPP

#pragma once

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

#include "enum/ServoType.hpp"

namespace behaviour_tree::node::custom::action
```

```
{
    class SetAngle final : public CustomNode
    {
    public:
        SetAngle(const std::string &name, const ServoType servo_type, const int angle) : CustomNode(name), servo_type(servo_type), angle(angle)
        {
        }

        const static tl::expected<std::shared_ptr<SetAngle>, std::string> parse(const pugi::xml_node &node, const int index, const std::string &name_attribute)
        {
            const tl::expected<ServoType, std::string> maybe_servo_type = [&]()
            {
                const std::string servo_type_attribute = node.attribute("servo_type").as_string();
                tl::expected<ServoType, std::string> result;
                switch (utils::hash(servo_type_attribute))
                {
                case utils::hash("FrontWheels"):
                {
                    result = ServoType::FrontWheels;
                    break;
                }
                case utils::hash("CameraServo1"):
                {
                    result = ServoType::CameraServo1;
                    break;
                }
                case utils::hash("CameraServo2"):
                {
                    result = ServoType::CameraServo2;
                    break;
                }
                default:
                {
                    result = tl::unexpected(fmt::format(R"(Invalid servo_type: ’{}’ | Action:SetAngle:[’{}’,{}])", servo_type_attribute, name_attribute, index));
                }
                };
                return result;
            }();
            const int angle = node.attribute("angle").as_int();
            if (angle < 0 || angle > 180)
            {
                return tl::unexpected(fmt::format(R"(Invalid angle: ’{}’ | Action:SetAngle:[’{}’,{}])", angle, name_attribute, index));
            }
            return std::make_shared<SetAngle>(SetAngle(name_attribute, maybe_servo_type.value(), angle));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();
            const MovementSystem *movement_system = car_system->getMovementSystem();
            switch (this->getServoType())
            {
            case ServoType::FrontWheels:
                movement_system->setFrontWheelsAngle(this->getAngle());
                break;
            case ServoType::CameraServo1:
                movement_system->setCameraServo1Angle(this->getAngle());
                break;
            case ServoType::CameraServo2:
                movement_system->setCameraServo2Angle(this->getAngle());
                break;
            }
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Success;
        }

        const std::string toString() const final override
        {
            std::string servo_type;
            switch (this->getServoType())
            {
            case ServoType::FrontWheels:
                servo_type = "FrontWheels";
                break;
            case ServoType::CameraServo1:
                servo_type = "CameraServo1";
                break;
            case ServoType::CameraServo2:
                servo_type = "CameraServo2";
                break;
            }
            const std::string &name = this->getName();
             if (name != "")
                 return fmt::format(R"(<Action:SetAngle name='{}' servo_type='{}' angle='{}'/>)", name,
      servo_type, this->getAngle());
             else
                 return fmt::format(R"(<Action:SetAngle servo_type='{}' angle='{}'/>)", servo_type,
      this->getAngle());
         }

         const ServoType getServoType() const
         {
             return this->servo_type;
         }

         const int getAngle() const
         {
             return this->angle;
         }

     private:
         const int angle;
         const ServoType servo_type;
     };
 }

 #endif
```

8.76 behaviour_tree_car/include/behaviour_tree/node/custom/action/↩ SetSpeed.hpp File Reference

```
#include <fmt/format.h>
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include "enum/WheelType.hpp"
```

Classes

- class behaviour_tree::node::custom::action::SetSpeed

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

8.77 SetSpeed.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SETSPEED_HPP
#define BEHAVIOUR_TREE_SETSPEED_HPP

#pragma once

#include <fmt/format.h>

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
```

```cpp
#include "behaviour_tree/CarContext.hpp"
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

#include "enum/WheelType.hpp"

namespace behaviour_tree::node::custom::action
{
    class SetSpeed final : public CustomNode
    {
    public:
        SetSpeed(const std::string &name, const WheelType wheel_type, const int speed) :
    CustomNode(name), wheel_type(wheel_type), speed(speed)
        {
        }

        const static tl::expected<std::shared_ptr<SetSpeed>, std::string> parse(const pugi::xml_node
    &node, const int index, const std::string &name_attribute)
        {
            const tl::expected<WheelType, std::string> maybe_wheel_type = [&]()
            {
                const std::string wheel_type_attribute = node.attribute("wheel_type").as_string();
                tl::expected<WheelType, std::string> result;
                switch (utils::hash(wheel_type_attribute))
                {
                case utils::hash("Left"):
                {
                    result = WheelType::Left;
                    break;
                }
                case utils::hash("Right"):
                {
                    result = WheelType::Right;
                    break;
                }
                case utils::hash("Both"):
                {
                    result = WheelType::Both;
                    break;
                }
                default:
                {
                    result = tl::unexpected(fmt::format(R"(Invalid wheel_type: '{}' |
    Action:SetSpeed:['{}',{}])", wheel_type_attribute, name_attribute, index));
                    break;
                }
                };
                return result;
            }();
            if (!maybe_wheel_type.has_value())
            {
                return tl::make_unexpected(maybe_wheel_type.error());
            }
            const int speed = node.attribute("speed").as_int();
            if (speed < 0 || speed > 100)
            {
                return tl::unexpected(fmt::format(R"(Invalid speed: '{}' | Action:SetSpeed:['{}',{}])",
    speed, name_attribute, index));
            }
            return std::make_shared<SetSpeed>(SetSpeed(name_attribute, maybe_wheel_type.value(), speed));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();
            const MovementSystem *movement_system = car_system->getMovementSystem();
            switch (this->getWheelType())
            {
            case WheelType::Left:
                movement_system->setRearLeftWheelSpeed(this->getSpeed());
                break;
            case WheelType::Right:
                movement_system->setRearRightWheelSpeed(this->getSpeed());
                break;
            case WheelType::Both:
                movement_system->setRearWheelsSpeed(this->getSpeed());
                break;
            }
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Success;
        }

        const std::string toString() const final override
        {
            std::string wheel_type;
            switch (this->getWheelType())
            {
            case WheelType::Left:
                wheel_type = "Forward";
                break;
            case WheelType::Right:
                 wheel_type = "Backward";
                 break;
             case WheelType::Both:
                 wheel_type = "Both";
                 break;
             }
             const std::string &name = this->getName();
             if (name != "")
                 return fmt::format(R"(<Action:SetSpeed name='{}' wheel_type='{}' speed='{}'/>)", name,
       wheel_type, this->getSpeed());
             else
                 return fmt::format(R"(<Action:SetSpeed wheel_type='{}' speed='{}'/>)", wheel_type,
       this->getSpeed());
         }

         const WheelType getWheelType() const
         {
             return this->wheel_type;
         }

         const int getSpeed() const
         {
             return this->speed;
         }

     private:
         const int speed;
         const WheelType wheel_type;
     };
 }

 #endif
```

8.78 behaviour_tree_car/include/behaviour_tree/node/custom/action/↩ SetWheelDirection.hpp File Reference

```
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include "enum/DirectionType.hpp"
#include "enum/WheelType.hpp"
```

Classes

- class behaviour_tree::node::custom::action::SetWheelDirection

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

8.79 SetWheelDirection.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SETWHEELDIRECTION_HPP
#define BEHAVIOUR_TREE_SETWHEELDIRECTION_HPP

#pragma once

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

#include "enum/DirectionType.hpp"
#include "enum/WheelType.hpp"

namespace behaviour_tree::node::custom::action
{
    class SetWheelDirection final : public CustomNode
    {
    public:
        SetWheelDirection(const std::string &name, const WheelType wheel_type, const DirectionType
    direction_type) : CustomNode(name), wheel_type(wheel_type), direction_type(direction_type)
        {
        }

        const static tl::expected<std::shared_ptr<SetWheelDirection>, std::string> parse(const
    pugi::xml_node &node, const int index, const std::string &name_attribute)
        {
            const tl::expected<WheelType, std::string> maybe_wheel_type = [&]()
            {
                const std::string wheel_type_attribute = node.attribute("wheel_type").as_string();
                tl::expected<WheelType, std::string> result;
                switch (utils::hash(wheel_type_attribute))
                {
                case utils::hash("Left"):
                {
                    result = WheelType::Left;
                    break;
                }
                case utils::hash("Right"):
                {
                    result = WheelType::Right;
                    break;
                }
                case utils::hash("Both"):
                {
                    result = WheelType::Both;
                    break;
                }
                default:
                {
                    result = tl::unexpected(fmt::format(R"(Invalid wheel_type: '{}' |
    Action:SetWheelDirection:['{}',{}])", wheel_type_attribute, name_attribute, index));
                    break;
                }
                };
                return result;
            }();
            if (!maybe_wheel_type.has_value())
            {
                return tl::make_unexpected(maybe_wheel_type.error());
            }
            const tl::expected<DirectionType, std::string> maybe_direction_type =
                [&]()
            {
                const std::string direction_type_attribute =
    node.attribute("direction_type").as_string();
                tl::expected<DirectionType, std::string> result;
                switch (utils::hash(direction_type_attribute))
                {
                case utils::hash("Forward"):
                {
                    result = DirectionType::Forward;
                }
                case utils::hash("Backward"):
                {
                    result = DirectionType::Backward;
                }
                default:
                {
                    result = tl::make_unexpected(fmt::format(R"(Invalid direction_type: '{}' |
    Action:Drive:['{}',{}])", direction_type_attribute, name_attribute, index));
```

```
                }
                };
                return result;
            }();
            if (!maybe_direction_type.has_value())
            {
                return tl::make_unexpected(maybe_direction_type.error());
            }
            return std::make_shared<SetWheelDirection>(SetWheelDirection(name_attribute,
       maybe_wheel_type.value(), maybe_direction_type.value()));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();
            const MovementSystem *movement_system = car_system->getMovementSystem();
            switch (this->getDirectionType())
            {
            case DirectionType::Forward:
            {
                switch (this->getWheelType())
                {
                case WheelType::Left:
                {
                    movement_system->setRearLeftWheelDirectionToForward();
                    break;
                }
                case WheelType::Right:
                {
                    movement_system->setRearRightWheelDirectionToForward();
                    break;
                }
                case WheelType::Both:
                {
                    movement_system->setRearWheelsDirectionToForward();
                    break;
                }
                }
                break;
            }
            case DirectionType::Backward:
            {
                switch (this->getWheelType())
                {
                case WheelType::Left:
                {
                    movement_system->setRearLeftWheelDirectionToBackward();
                    break;
                }
                case WheelType::Right:
                {
                    movement_system->setRearRightWheelDirectionToBackward();
                    break;
                }
                case WheelType::Both:
                {
                    movement_system->setRearWheelsDirectionToBackward();
                    break;
                }
                }
                break;
            }
            }
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Success;
        }

        const std::string toString() const final override
        {
            std::string direction_type;
            switch (this->getDirectionType())
            {
            case DirectionType::Forward:
                direction_type = "Forward";
                break;
            case DirectionType::Backward:
                direction_type = "Backward";
                break;
            }
            std::string wheel_type;
            switch (this->getWheelType())
            {
            case WheelType::Left:
                wheel_type = "Forward";
                break;
            case WheelType::Right:
                wheel_type = "Backward";
                break;
            case WheelType::Both:
                wheel_type = "Both";
                break;
            }
            const std::string &name = this->getName();
            if (name != "")
                return fmt::format(R"(<Action:SetWheelDirection name='{}' direction_type='{}' wheel_type='{}'/>)", name, direction_type, wheel_type);
            else
                return fmt::format(R"(<Action:SetWheelDirection direction_type='{}' wheel_type='{}'/>)", direction_type, wheel_type);
        }

        const DirectionType getDirectionType() const
        {
            return this->direction_type;
        }

        const WheelType getWheelType() const
        {
            return this->wheel_type;
        }

    private:
        const DirectionType direction_type;
        const WheelType wheel_type;
    };
}

#endif
```

8.80 behaviour_tree_car/include/behaviour_tree/node/custom/action/Turn.hpp File Reference

```
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
```

Classes

- class behaviour_tree::node::custom::action::Turn

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::action

8.81 Turn.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_TURN_HPP
#define BEHAVIOUR_TREE_TURN_HPP

#pragma once

#include "behaviour_tree/node/custom/CustomNode.hpp"

```

```
#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#endif // !BEHAVIOUR_TREE_DISABLE_RUN


namespace behaviour_tree::node::custom::action
{
    class Turn final : public CustomNode
    {
    public:
        Turn(const std::string &name, const int angle) : CustomNode(name), angle(angle)
        {
        }

        const static tl::expected<std::shared_ptr<Turn>, std::string> parse(const pugi::xml_node &node,
    const int index, const std::string &name_attribute)
        {
            const int angle = node.attribute("angle").as_int();
            if (angle < 0 || angle > 180)
            {
                return tl::unexpected(fmt::format(R"(Invalid angle: '{}' | Action:SetAngle:['{}',{}])",
    angle, name_attribute, index));
            }
            return std::make_shared<Turn>(Turn(name_attribute, angle));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();
            const MovementSystem *movement_system = car_system->getMovementSystem();
            movement_system->setFrontWheelsAngle(this->getAngle());
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Success;
        }

        const std::string toString() const final override
        {
            const std::string &name = this->getName();
            if (name != "")
                return fmt::format(R"(<Action:Turn name='{}' angle='{}'/>)", name, this->getAngle());
            else
                return fmt::format(R"(<Action:Turn angle='{}'/>)", this->getAngle());
        }

        const int getAngle() const
        {
            return this->angle;
        }

    private:
        const int angle;
    };
}

#endif
```

8.82 behaviour_tree_car/include/behaviour_tree/node/custom/CarCustomNodeParser.hpp File Reference

```
#include <string>
#include <memory>
#include <tl/expected.hpp>
#include <pugixml.hpp>
#include <fmt/format.h>
#include "behaviour_tree/BehaviourTree.hpp"
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/node/custom/CustomNodeParser.hpp"
#include "action/PauseExecution.hpp"
#include "action/Print.hpp"
#include "action/Drive.hpp"
#include "action/Turn.hpp"
```

```
#include "action/SetSpeed.hpp"
#include "action/SetWheelDirection.hpp"
#include "action/SetAngle.hpp"
#include "condition/SucceedOnAverageNearbyScan.hpp"
#include "condition/SucceedOnAnyNearbyScan.hpp"
#include "condition/SucceedOnAverageColour.hpp"
#include "utils/Utility.hpp"
```

Classes

- class behaviour_tree::node::custom::CarCustomNodeParser

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom

8.83 CarCustomNodeParser.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_CAR_TASK_NODE_PARSER_HPP
#define BEHAVIOUR_TREE_CAR_TASK_NODE_PARSER_HPP

#pragma once

#include <string>
#include <memory>

#include <tl/expected.hpp>
#include <pugixml.hpp>
#include <fmt/format.h>

#include "behaviour_tree/BehaviourTree.hpp"
#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/node/custom/CustomNodeParser.hpp"

#include "action/PauseExecution.hpp"

#include "action/Print.hpp"

#include "action/Drive.hpp"
#include "action/Turn.hpp"
#include "action/SetSpeed.hpp"
#include "action/SetWheelDirection.hpp"
#include "action/SetAngle.hpp"

#include "condition/SucceedOnAverageNearbyScan.hpp"
#include "condition/SucceedOnAnyNearbyScan.hpp"
#include "condition/SucceedOnAverageColour.hpp"

#include "utils/Utility.hpp"

namespace behaviour_tree::node::custom
{
    class CarCustomNodeParser : public custom::CustomNodeParser
    {
    public:
        CarCustomNodeParser() {}

        tl::expected<std::shared_ptr<custom::CustomNode>, std::string> parseCustomNode(const
    pugi::xml_node &node, const int index) override
        {
            const std::string name_attribute = node.attribute("node_name").as_string();
            const std::string node_name = node.name();
            switch (utils::hash(node_name))
            {
            case utils::hash("Action:PauseExecution"):
                return custom::action::PauseExecution::parse(node, index, name_attribute);
            case utils::hash("Action:Print"):
                return custom::action::Print::parse(node, index, name_attribute);
            case utils::hash("Action:Drive"):
                return custom::action::Drive::parse(node, index, name_attribute);
            case utils::hash("Action:Turn"):
                return custom::action::Turn::parse(node, index, name_attribute);
            case utils::hash("Action:SetSpeed"):
                return custom::action::SetSpeed::parse(node, index, name_attribute);
            case utils::hash("Action:SetAngle"):
                return custom::action::SetAngle::parse(node, index, name_attribute);
            case utils::hash("Action:SetWheelDirection"):
                return custom::action::SetWheelDirection::parse(node, index, name_attribute);
            case utils::hash("Condition:SucceedOnAverageNearbyScan"):
                return custom::condition::SucceedOnAverageNearbyScan::parse(node, index, name_attribute);
            case utils::hash("Condition:SucceedOnAnyNearbyScan"):
                return custom::condition::SucceedOnAnyNearbyScan::parse(node, index, name_attribute);
            case utils::hash("Condition:SucceedOnAverageColour"):
                return custom::condition::SucceedOnAverageColour::parse(node, index, name_attribute);
            default:
            {
                return tl::unexpected(fmt::format(R"(Invalid custom node type: '{}' | {}:['{}',{}])",
       node_name, node_name, name_attribute, index));
            }
            }
        }
    };
}

#endif
```

8.84 behaviour_tree_car/include/behaviour_tree/node/custom/condition/SucceedOnAnyNearbyScan.hpp File Reference

```
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
#include "behaviour_tree/global/distance.h"
```

Classes

- class behaviour_tree::node::custom::condition::SucceedOnAnyNearbyScan

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::condition

8.85 SucceedOnAnyNearbyScan.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_ANY_SCAN_HPP
#define BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_ANY_SCAN_HPP

#pragma once

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

#include "behaviour_tree/global/distance.h"

namespace behaviour_tree::node::custom::condition
{
    class SucceedOnAnyNearbyScan final : public CustomNode
    {
    public:
        SucceedOnAnyNearbyScan(const std::string &name, const int min_angle, const int max_angle, const double cm) : CustomNode(name),
                  min_angle(min_angle),
                  max_angle(max_angle),
                  cm(cm),
                  minimum_distance_unit(cm * global::CM_TO_DISTANCE)
        {
        }

        const static tl::expected<std::shared_ptr<SucceedOnAnyNearbyScan>, std::string> parse(const pugi::xml_node &node, const int index, const std::string &name_attribute)
        {
            const int min_angle = node.attribute("min_angle").as_int();
            if (min_angle < 0 || min_angle > 360)
            {
                return tl::unexpected(fmt::format(R"(Invalid min_angle: '{}' | Condition:SucceedOnAnyNearbyScan:['{}',{}])", min_angle, name_attribute, index));
            }
            const int max_angle = node.attribute("max_angle").as_int();
            if (max_angle < 0 || max_angle > 360)
            {
                return tl::unexpected(fmt::format(R"(Invalid max_angle: '{}' | Condition:SucceedOnAnyNearbyScan:['{}',{}])", max_angle, name_attribute, index));
            }
            const double cm = node.attribute("cm").as_double();
            if (cm < 0)
            {
                return tl::unexpected(fmt::format(R"(Invalid cm: '{}' | Condition:SucceedOnAnyNearbyScan:['{}',{}])", cm, name_attribute, index));
            }
            return std::make_shared<SucceedOnAnyNearbyScan>(
                SucceedOnAnyNearbyScan(
                    name_attribute,
                    min_angle,
                    max_angle,
                    cm));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();
            std::vector<Measure> scan_data = car_system->getDeviceManager()->getLidarDevice()->getScanData();
            for (auto &measure : scan_data)
            {
                if (measure.angle > this->getMinAngle() && measure.angle < this->getMaxAngle())
                {
                    if (measure.distance < this->getMinimumDistanceUnit())
                    {
                        return Status::Success;
                    }
                }
            }
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Failure;
        }

        const int getMinAngle() const
        {
            return this->min_angle;
        }

        const int getMaxAngle() const
        {
            return this->max_angle;
        }

        const double getMinimumDistanceUnit() const
        {
            return this->minimum_distance_unit;
        }

        const double getCentimeters() const
        {
            return this->cm;
        }

        const std::string toString() const final override
        {
            const std::string &name = this->getName();
            if (name != "")
                return fmt::format(R"(<Condition:SucceedOnAnyNearbyScan name='{}' min_angle='{}'
      max_angle='{}' cm='{}'/>)", name, this->getMinAngle(), this->getMaxAngle(), this->getCentimeters());
            else
                 return fmt::format(R"(<Condition:SucceedOnAnyNearbyScan min_angle='{}' max_angle='{}'
      cm='{}'/>)", this->getMinAngle(), this->getMaxAngle(), this->getCentimeters());
         }

     private:
         const int min_angle;
         const int max_angle;

         const double cm;

         const double minimum_distance_unit;
     };
}

#endif
```

8.86 behaviour_tree_car/include/behaviour_tree/node/custom/condition/SucceedOnAverageColour.hpp File Reference

```
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
#include <opencv2/opencv.hpp>
```

Classes

- class behaviour_tree::node::custom::condition::SucceedOnAverageColour

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::condition

8.87 SucceedOnAverageColour.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_COLOUR_HPP
#define BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_COLOUR_HPP

#pragma once

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
#include <opencv2/opencv.hpp>
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

namespace behaviour_tree::node::custom::condition
{
    class SucceedOnAverageColour final : public CustomNode
    {
    public:
        SucceedOnAverageColour(const std::string &name, const std::string hex_colour, const double tolerance) : CustomNode(name),
    hex_colour(hex_colour),
    tolerance(tolerance)
        {
        }

        const static tl::expected<std::shared_ptr<SucceedOnAverageColour>, std::string> parse(const pugi::xml_node &node, const int index, const std::string &name_attribute)
        {
            std::string hex_colour = node.attribute("hex_colour").as_string();
            if (!hex_colour.compare(0, 1, "#"))
            {
                hex_colour = "#" + hex_colour;
            }
            if (hex_colour.size() != 7)
            {
                return tl::unexpected(fmt::format(R"(Invalid hex_colour: '{}' | Condition:SucceedOnAverageColour:['{}',{}])", hex_colour, name_attribute, index));
            }
            for (int i = 1; i < hex_colour.size(); i++)
            {
                if (!std::isxdigit(hex_colour[i]))
                {
                    return tl::unexpected(fmt::format(R"(Invalid hex_colour: '{}' | Condition:SucceedOnAverageColour:['{}',{}])", hex_colour, name_attribute, index));
                }
            }
            const double tolerance = node.attribute("tolerance").as_double();
            if (tolerance < 0 || tolerance > 100)
            {
                return tl::unexpected(fmt::format(R"(Invalid tolerance: '{}' | Condition:SucceedOnAverageColour:['{}',{}])", tolerance, name_attribute, index));
            }
            return std::make_shared<SucceedOnAverageColour>(
                SucceedOnAverageColour(
                    name_attribute,
                    hex_colour,
                    tolerance));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
            // Following made by ChatGPT
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();

            std::string frame_buffer = car_system->getDeviceManager()->getCameraDevice()->getFrameBuffer();
            std::vector<uchar> buffer(frame_buffer.begin(), frame_buffer.end());
            cv::Mat frame = cv::imdecode(buffer, cv::IMREAD_COLOR);
            if (frame.empty())
            {
                spdlog::error("Failed to decode frame buffer");
                return Status::Failure;
            }

            cv::Scalar avg_color = cv::mean(frame);
            std::string avg_color_hex = fmt::format("#{:02x}{:02x}{:02x}", static_cast<int>(avg_color[2]), static_cast<int>(avg_color[1]), static_cast<int>(avg_color[0]));

            int color_diff = calculateColorDifference(avg_color_hex, this->hex_colour);
            int max_color_diff = calculateMaxColorDifference(this->tolerance);

            if (color_diff <= max_color_diff)
            {
                return Status::Success;
            }
            else
            {
                return Status::Failure;
            }
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Failure;
        }

        // Following made by ChatGPT
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
        // Function to calculate the color difference between two hex colors
        int calculateColorDifference(const std::string &color1, const std::string &color2)
        {
            // Convert hex strings to RGB values
            cv::Scalar rgb_color1 = hexToRGB(color1);
            cv::Scalar rgb_color2 = hexToRGB(color2);

            // Calculate the Euclidean distance between the two colors in RGB space
            return static_cast<int>(cv::norm(rgb_color1, rgb_color2));
        }

        // Function to convert a hex color string to RGB values
        cv::Scalar hexToRGB(const std::string &hexColor)
        {
            std::string hex = hexColor.substr(1); // remove the '#' character
            int r, g, b;
            std::istringstream(hex.substr(0, 2)) » std::hex » r;
            std::istringstream(hex.substr(2, 2)) » std::hex » g;
            std::istringstream(hex.substr(4, 2)) » std::hex » b;
            return cv::Scalar(b, g, r);
        }

        // Function to calculate the maximum allowable color difference based on the percentage
    threshold
        int calculateMaxColorDifference(double percentage)
        {
            // Assuming percentage represents the acceptable deviation from the specified color
            // We can calculate the maximum allowable color difference based on this percentage
            // For example, if percentage is 10, it means 10% deviation is acceptable

            // Max color difference will be based on the maximum possible Euclidean distance between two
    colors in RGB space
            // For simplicity, let's assume the maximum distance is between pure black and pure white
            cv::Scalar black(0, 0, 0);
            cv::Scalar white(255, 255, 255);
            int max_color_diff = static_cast<int>(cv::norm(black, white));

            // Now, we scale the maximum color difference based on the percentage
            return static_cast<int>(max_color_diff * (percentage / 100.0));
        }
#endif // !BEHAVIOUR_TREE_DISABLE_RUNss

        const std::string getHexColour() const
        {
            return this->hex_colour;
        }

        const double getTolerance() const
        {
            return this->tolerance;
        }

        const std::string toString() const final override
        {
            const std::string &name = this->getName();
            if (name != "")
                return fmt::format(R"(<Condition:SucceedOnAverageColour name='{}' hex_colour='{}'
    tolerance='{}'/>)", name, this->getHexColour(), this->getTolerance());
            else
                return fmt::format(R"(<Condition:SucceedOnAverageColour hex_colour='{}'
    tolerance='{}'/>)", this->getHexColour(), this->getTolerance());
        }

    private:
        const std::string hex_colour;
        const double tolerance;
    };
}

```

```
#endif
```

8.88 behaviour_tree_car/include/behaviour_tree/node/custom/condition/SucceedOnAverageNearbyScan.hpp File Reference

```
#include "behaviour_tree/node/custom/CustomNode.hpp"
#include "behaviour_tree/Context.h"
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
#include "behaviour_tree/global/distance.h"
```

Classes

- class behaviour_tree::node::custom::condition::SucceedOnAverageNearbyScan

Namespaces

- namespace behaviour_tree
- namespace behaviour_tree::node
- namespace behaviour_tree::node::custom
- namespace behaviour_tree::node::custom::condition

8.89 SucceedOnAverageNearbyScan.hpp

Go to the documentation of this file.

```
#ifndef BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_NEARBY_SCAN_HPP
#define BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_NEARBY_SCAN_HPP

#pragma once

#include "behaviour_tree/node/custom/CustomNode.hpp"

#include "behaviour_tree/Context.h"
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
#include "behaviour_tree/CarContext.hpp"
#include <spdlog/spdlog.h>
#endif // !BEHAVIOUR_TREE_DISABLE_RUN

#include "behaviour_tree/global/distance.h"

namespace behaviour_tree::node::custom::condition
{
    class SucceedOnAverageNearbyScan final : public CustomNode
    {
    public:
        SucceedOnAverageNearbyScan(const std::string &name, const int min_angle, const int max_angle,
    const double cm, const int smallest_measure_amount_used) : CustomNode(name),

                                                        min_angle(min_angle),

                                                        max_angle(max_angle),

                                                        cm(cm),

    smallest_measure_amount_used(smallest_measure_amount_used),
```

```cpp
                                                           average_distance_unit(cm * global::CM_TO_DISTANCE)
        {
        }

        const static tl::expected<std::shared_ptr<SucceedOnAverageNearbyScan>, std::string> parse(const pugi::xml_node &node, const int index, const std::string &name_attribute)
        {
            const int min_angle = node.attribute("min_angle").as_int();
            if (min_angle < 0 || min_angle > 360)
            {
                return tl::unexpected(fmt::format(R"(Invalid min_angle: '{}' | Condition:SucceedOnAverageNearbyScan:['{}',{}])", min_angle, name_attribute, index));
            }
            const int max_angle = node.attribute("max_angle").as_int();
            if (max_angle < 0 || max_angle > 360)
            {
                return tl::unexpected(fmt::format(R"(Invalid max_angle: '{}' | Condition:SucceedOnAverageNearbyScan:['{}',{}])", max_angle, name_attribute, index));
            }
            const double cm = node.attribute("cm").as_double();
            if (cm < 0)
            {
                return tl::unexpected(fmt::format(R"(Invalid cm: '{}' | Condition:SucceedOnAverageNearbyScan:['{}',{}])", cm, name_attribute, index));
            }
            const int minimum_measure_amount_used = node.attribute("minimum_measure_amount_used").as_int();
            if (minimum_measure_amount_used < 0)
            {
                return tl::unexpected(fmt::format(R"(Invalid minimum_measure_amount_used: '{}' | Condition:SucceedOnAverageNearbyScan:['{}',{}])", minimum_measure_amount_used, name_attribute, index));
            }
            return std::make_shared<SucceedOnAverageNearbyScan>(
                SucceedOnAverageNearbyScan(
                    name_attribute,
                    min_angle,
                    max_angle,
                    cm,
                    minimum_measure_amount_used));
        }

        const Status run(const int tick_count, std::shared_ptr<Context> context) final override
        {
#ifndef BEHAVIOUR_TREE_DISABLE_RUN
            std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
            auto car_system = car_context->getCarSystem();
            double total_distance = 0.0;
            int angles_between_count = 0;
            std::vector<Measure> scan_data = car_system->getDeviceManager()->getLidarDevice()->getScanData();
            if (this->smallest_measure_amount_used > 0)
            {
                std::vector<Measure> smallest_measures;
                std::partial_sort(scan_data.begin(), scan_data.begin() + this->smallest_measure_amount_used, scan_data.end(),
                                  [](const Measure &a, const Measure &b)
                                  { return a.distance < b.distance; });
                smallest_measures.insert(smallest_measures.end(), scan_data.begin(), scan_data.begin() + this->smallest_measure_amount_used);
                scan_data = smallest_measures;
            }
            for (auto &measure : scan_data)
            {
                if (measure.angle > this->getMinAngle() && measure.angle < this->getMaxAngle())
                {
                    total_distance += measure.distance;
                    ++angles_between_count;
                }
            }
            if (angles_between_count > 0)
            {
                double average_distance_unit = total_distance / angles_between_count;
                if (average_distance_unit < this->getAverageDistanceUnit())
                {
                    return Status::Success;
                }
            }
#endif // !BEHAVIOUR_TREE_DISABLE_RUN
            return Status::Failure;
        }

        const int getMinAngle() const
        {
             return this->min_angle;
        }

        const int getMaxAngle() const
        {
            return this->max_angle;
        }

        const double getAverageDistanceUnit() const
        {
            return this->average_distance_unit;
        }

        const double getCentimeters() const
        {
            return this->cm;
        }

        const int getSmallestMeasureAmountUsed() const
        {
            return this->smallest_measure_amount_used;
        }

        const std::string toString() const final override
        {
            const std::string &name = this->getName();
            if (name != "")
                return fmt::format(R"(<Condition:SucceedOnAverageNearbyScan name='{}' min_angle='{}'
      max_angle='{}' cm='{}' smallest_measure_amount_used='{}'/>)", name, this->getMinAngle(),
      this->getMaxAngle(), this->getCentimeters(), this->getSmallestMeasureAmountUsed());
            else
                return fmt::format(R"(<Condition:SucceedOnAverageNearbyScan min_angle='{}'
      max_angle='{}' cm='{}' smallest_measure_amount_used='{}'/>)", this->getMinAngle(),
      this->getMaxAngle(), this->getCentimeters(), this->getSmallestMeasureAmountUsed());
        }

    private:
        const int min_angle;
        const int max_angle;

        const double cm;

        const int smallest_measure_amount_used;

        const double average_distance_unit;
    };
}

#endif
```

8.90 behaviour_tree_tester/src/main.cpp File Reference

```
#include <iostream>
#include <chrono>
#include <filesystem>
#include <thread>
#include <memory>
#include <cxxopts.hpp>
#include "behaviour_tree/BehaviourTreeParser.hpp"
#include "behaviour_tree/node/custom/CarCustomNodeParser.hpp"
```

Functions

- int main (int argc, const char ∗argv[ ])

8.90.1 Function Documentation

8.90.1.1 main()

```
int main (
            int argc,
            const char * argv[] )
```

8.91 behaviour_tree_validator/src/main.cpp File Reference

```
#include <iostream>
#include <cxxopts.hpp>
#include <rapidjson/document.h>
#include <rapidjson/writer.h>
#include <rapidjson/stringbuffer.h>
#include "behaviour_tree/BehaviourTreeParser.hpp"
#include "behaviour_tree/node/custom/CarCustomNodeParser.hpp"
```

Functions

- int main (int argc, char *argv[ ])

8.91.1 Function Documentation

8.91.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

8.92 README.md File Reference

8.93 rplidar/include/RPLidar.h File Reference

```
#include <string>
#include <stdint.h>
#include <serial/serial.h>
#include <iostream>
#include <map>
#include <vector>
#include <chrono>
#include <thread>
#include <sstream>
#include <iomanip>
#include <functional>
#include <tuple>
#include <stdexcept>
#include <memory>
#include <spdlog/spdlog.h>
#include <spdlog/fmt/bin_to_hex.h>
#include <tl/expected.hpp>
```

Classes

- struct rplidar::ScanInfo
- struct rplidar::DeviceInfo
- struct rplidar::HealthInfo

  Health Info for Lidar Scanner.
- struct rplidar::Measure
- class rplidar::ExpressPacket
- class rplidar::RPLidar

  Class for communicating with RPLidar rangefinder scanners.

Namespaces

- namespace rplidar

Enumerations

- enum rplidar::ScanType { rplidar::NORMAL = 0 , rplidar::FORCE , rplidar::EXPRESS }

Variables

- constexpr uint8_t rplidar::SYNC_BYTE = 0xA5
- constexpr uint8_t rplidar::SYNC_BYTE2 = 0x5A
- constexpr uint8_t rplidar::GET_INFO_BYTE = 0x50
- constexpr uint8_t rplidar::GET_HEALTH_BYTE = 0x52
- constexpr uint8_t rplidar::STOP_BYTE = 0x25
- constexpr uint8_t rplidar::RESET_BYTE = 0x40
- constexpr int rplidar::DESCRIPTOR_LEN = 7
- constexpr int rplidar::INFO_LEN = 20
- constexpr int rplidar::HEALTH_LEN = 3
- constexpr int rplidar::INFO_TYPE = 4
- constexpr int rplidar::HEALTH_TYPE = 6
- constexpr int rplidar::MAX_MOTOR_PWM = 1023
- constexpr int rplidar::DEFAULT_MOTOR_PWM = 660
- constexpr uint8_t rplidar::SET_PWM_BYTE = 0xF0

8.94 RPLidar.h

Go to the documentation of this file.
```
#ifndef RPLIDAR_H
#define RPLIDAR_H

#pragma once

// Made with the help of ChatGPT


/*
#include <RPLidar.h>
#include <memory>
#include <spdlog/spdlog.h>

int main()
{
    using namespace rplidar;
    spdlog::set_level(spdlog::level::debug);

    // auto lidar_result = RPLidar::create("/dev/ttyUSB0"); // For Linux
    // auto lidar_result = RPLidar::create("COM3"); // For Windows
    if (!lidar_result.has_value())
    {
        std::cout « "Unable to open lidar\n";
        return 0;
    }
    auto &lidar = lidar_result.value();
    std::cout « "Connected to lidar\n";

    auto info_result = lidar->get_info();
    if (!info_result.has_value())
    {
        std::cout « "Unable to get value of get_info() " « info_result.error();
        return 0;
    }
    auto &info = info_result.value();
    std::cout « fmt::format("model: {}, firmware: ({}, {}), hardware: {}, serialnumber: {}\n",
    info.model, info.firmware.first, info.firmware.second, info.hardware, info.serialNumber);

    auto health_result = lidar->get_health();
    if (!health_result.has_value())
    {
        std::cout « "Unable to get value of get_health() "« health_result.error();
        return 0;
    }
    auto &health = health_result.value();
    std::cout « fmt::format("({}, {})\n", health.status, health.errorCode);

    std::function<std::vector<Measure>()> scanGenerator = lidar->iter_scans();
    for (int i = 0; i < 10; i++)
    {
        std::vector<Measure> scan = scanGenerator();
        std::cout « "Got " « scan.size() « " Measures!\n";
        for (const Measure &measure : scan)
        {
            // Access individual measurements in the scan
            bool newScan = measure.newScan;
            int quality = measure.quality;
            float angle = measure.angle;
            float distance = measure.distance;
        }
    }

    lidar->stop();
    lidar->stop_motor();
    lidar->disconnect();
    return 1;
}

*/

#include <string>
#include <stdint.h>

#include <serial/serial.h>

#include <iostream>
#include <map>
#include <vector>

// std::this_thread::sleep_for(std::chrono::milliseconds(1));
#include <chrono>
#include <thread>

// convertToHexString
#include <sstream>
#include <iomanip>

// iter_measures
#include <functional>

#include <tuple>
#include <stdexcept>

#include <memory>

#include <spdlog/spdlog.h>
#include <spdlog/fmt/bin_to_hex.h>

#include <tl/expected.hpp>

namespace rplidar
{
    constexpr uint8_t SYNC_BYTE = 0xA5;
    constexpr uint8_t SYNC_BYTE2 = 0x5A;
```

```cpp

constexpr uint8_t GET_INFO_BYTE = 0x50;
constexpr uint8_t GET_HEALTH_BYTE = 0x52;

constexpr uint8_t STOP_BYTE = 0x25;
constexpr uint8_t RESET_BYTE = 0x40;

enum ScanType
{
    NORMAL = 0,
    FORCE,
    EXPRESS
};

struct ScanInfo
{
    int currently_scanning;
    int dsize;
    ScanType type;
};

static std::map<ScanType, std::map<std::string, uint8_t» SCAN_TYPE = {
    {ScanType::NORMAL, {{"byte", 0x20}, {"response", 129}, {"size", 5}}},
    {ScanType::FORCE, {{"byte", 0x21}, {"response", 129}, {"size", 5}}},
    {ScanType::EXPRESS, {{"byte", 0x82}, {"response", 130}, {"size", 84}}}};

constexpr int DESCRIPTOR_LEN = 7;
constexpr int INFO_LEN = 20;
constexpr int HEALTH_LEN = 3;

constexpr int INFO_TYPE = 4;
constexpr int HEALTH_TYPE = 6;

constexpr int MAX_MOTOR_PWM = 1023;
constexpr int DEFAULT_MOTOR_PWM = 660;
constexpr uint8_t SET_PWM_BYTE = 0xF0;

static std::map<int, std::string> HEALTH_STATUSES = {
    {0, "Good"},
    {1, "Warning"},
    {2, "Error"}};

struct DeviceInfo
{
    uint8_t model;
    std::pair<uint8_t, uint8_t> firmware;
    uint8_t hardware;
    std::string serialNumber;
};

struct HealthInfo
{
    std::string status;
    int errorCode;
};

struct Measure
{
    bool newScan;
    int quality;
    double angle;
    double distance;
};

class ExpressPacket
{
public:
    static const uint8_t sync1 = 0xa;
    static const uint8_t sync2 = 0x5;

    ExpressPacket(std::vector<uint8_t> data)
    {
        if ((data[0] » 4) != sync1 || (data[1] » 4) != sync2)
        {
            throw std::invalid_argument("try to parse corrupted data");
        }

        uint8_t checksum = 0;
        for (size_t i = 2; i < data.size(); i++)
        {
            checksum ^= data[i];
        }

        if (checksum != ((data[0] & 0x0F) + ((data[1] & 0x0F) « 4)))
        {
            throw std::invalid_argument("Invalid checksum");
        }
```

```

            new_scan = (data[2] » 7) & 0x01;
            start_angle = static_cast<float>((data[1] & 0x0F) « 8 | data[2]) / 64.0f;

            for (size_t i = 4; i < data.size(); i += 5)
            {
                distance.push_back(((data[i + 1] » 2) & 0x3F) | ((data[i] & 0x3F) « 6));
                angle.push_back((((data[i + 3] & 0x0F) + ((data[i + 1] & 0x01) « 4)) / 8.0f) *
        getSign(data[i + 1]));
                distance.push_back(((data[i + 2] » 2) & 0x3F) | ((data[i + 1] & 0x3F) « 6));
                angle.push_back((((data[i + 3] » 4) & 0x0F) + ((data[i + 2] & 0x01) « 4)) / 8.0f *
        getSign(data[i + 2]));
            }
        }

        static int getSign(uint8_t value)
        {
            return (value & 0x02) ? -1 : 1;
        }

    public:
        std::vector<uint16_t> distance;
        std::vector<float> angle;
        bool new_scan;
        float start_angle;
    };
    class RPLidar
    {
    public:
        static tl::expected<std::unique_ptr<RPLidar>, std::string> create(const std::string &port,
        uint32_t baudrate = 115200U) noexcept
        {
            try
            {
                // Timeout after 60 seconds if nothing read
                auto timeout = serial::Timeout(1000U, 60000U, 0, 1000U, 0);
                std::unique_ptr<serial::Serial> serial = std::make_unique<serial::Serial>(port,
        baudrate, timeout);
                std::unique_ptr<RPLidar> lidar = std::make_unique<RPLidar>(port, baudrate,
        std::move(serial));
                return std::move(lidar);
            }
            catch (std::exception &e)
            {
                return tl::make_unexpected(e.what());
            }
        }

        RPLidar(const std::string &port, uint32_t baudrate, std::unique_ptr<serial::Serial> serial) :
        port(port), baudrate(baudrate), _serial(std::move(serial)){};

        ~RPLidar() {}

        void disconnect()
        {
            if (!this->_serial->isOpen())
            {
                return;
            }
            this->_serial->close();
        }

        void set_motor_speed(int pwm)
        {
            if (0 <= pwm && pwm <= MAX_MOTOR_PWM)
            {
                std::abort();
            }
            this->_motor_speed = pwm;
            if (this->motor_running)
                this->_set_pwm(this->_motor_speed);
        }

        void start_motor()
        {
            spdlog::info("Starting motor");
            // For A1
            this->_serial->setDTR(false);

            // For A2
            this->_set_pwm(this->_motor_speed);
            this->motor_running = true;
        }

        void stop_motor()
        {
            spdlog::info("Stopping motor");
            // For A2
            this->_set_pwm(0);
            std::this_thread::sleep_for(std::chrono::milliseconds(1));

            // For A1
            this->_serial->setDTR(true);
            this->motor_running = false;
        }

        tl::expected<DeviceInfo, std::string> get_info()
        {
            // Check if there's data in the buffer
            if (this->_serial->available() > 0)
            {
                throw std::runtime_error("Data in buffer, you can't have info! Run flush() to empty the buffer.");
            }
            this->_send_cmd(GET_INFO_BYTE);

            uint8_t dsize;
            bool isSingle;
            uint8_t dtype;
            auto descriptor_result = this->_read_descriptor();
            if (!descriptor_result.has_value())
                return tl::make_unexpected(descriptor_result.error());
            std::tie(dsize, isSingle, dtype) = descriptor_result.value();

            // Check response properties
            if (dsize != INFO_LEN)
            {
                return tl::make_unexpected("Wrong get_info reply length");
            }
            if (!isSingle)
            {
                return tl::make_unexpected("Not a single response mode");
            }
            if (dtype != INFO_TYPE)
            {
                return tl::make_unexpected("Wrong response data type");
            }

            // Read the response
            std::vector<uint8_t> raw = this->_read_response(dsize);

            // Convert serial number to a hex string
            std::string serialNumber;
            for (size_t i = 4; i < raw.size(); ++i)
            {
                serialNumber += convertToHexString(raw[i]);
            }

            // Construct the device info struct
            DeviceInfo info;
            info.model = raw[0];
            info.firmware.first = raw[2];
            info.firmware.second = raw[1];
            info.hardware = raw[3];
            info.serialNumber = serialNumber;

            return info;
        }

        tl::expected<HealthInfo, std::string> get_health()
        {
            // Check if there's data in the buffer
            if (this->_serial->available() > 0)
            {
                return tl::make_unexpected("Data in buffer, you can't get health info! Run cleanInput() to empty the buffer.");
            }
            spdlog::info("Asking for health");

            this->_send_cmd(GET_HEALTH_BYTE);

            // Read the descriptor
            uint8_t dsize;
            bool isSingle;
            uint8_t dtype;
            auto descriptor_result = this->_read_descriptor();
            if (!descriptor_result.has_value())
                return tl::make_unexpected(descriptor_result.error());
            std::tie(dsize, isSingle, dtype) = descriptor_result.value();

            // Check response properties
            if (dsize != HEALTH_LEN)
            {
                return tl::make_unexpected("Wrong get_health reply length");
            }
            if (!isSingle)
            {
                return tl::make_unexpected("Not a single response mode");
            }
            if (dtype != HEALTH_TYPE)
            {
                return tl::make_unexpected("Wrong response data type");
            }

            // Read the response
            std::vector<uint8_t> raw = this->_read_response(dsize);

            // Extract status and error code
            std::string status = HEALTH_STATUSES[raw[0]];
            int errorCode = (static_cast<int>(raw[1]) « 8) + static_cast<int>(raw[2]);

            return HealthInfo{status, errorCode};
        }

        void clean_input()
        {
            if (this->scanning.currently_scanning)
            {
                throw std::runtime_error("Cleaning not allowed during scanning process active!");
            }
            this->_serial->flushInput();
            this->express_trame = 32;
            this->express_data = nullptr;
        }

        void stop()
        {
            spdlog::info("Stopping scanning");
            this->_send_cmd(STOP_BYTE);
            std::this_thread::sleep_for(std::chrono::milliseconds(100));
            this->scanning.currently_scanning = false;
            this->clean_input();
        }

        tl::expected<nullptr_t, std::string> start(ScanType scanType = ScanType::NORMAL)
        {
            if (this->scanning.currently_scanning)
            {
                return tl::make_unexpected("Scanning already running!");
            }

            auto health_info_result = this->get_health();
            if (!health_info_result.has_value())
                return tl::make_unexpected(health_info_result.error());

            HealthInfo healthInfo = health_info_result.value();
            std::string status = healthInfo.status;
            int errorCode = healthInfo.errorCode;
            spdlog::debug("Health status: {} [{}]", status, errorCode);

            if (status == "Error")
            {
                this->reset();
                health_info_result = this->get_health();
                if (!health_info_result.has_value())
                    return tl::make_unexpected(health_info_result.error());
                healthInfo = health_info_result.value();
                status = healthInfo.status;
                errorCode = healthInfo.errorCode;
                if (status == "Error")
                {
                    return tl::make_unexpected("RPLidar hardware failure. Error code: " +
std::to_string(errorCode));
                }
            }
            else if (status == "Warning")
            {
                spdlog::warn("Warning sensor status detected! Error code: {}", errorCode);
            }

            uint8_t cmd = SCAN_TYPE[scanType]["byte"];

            spdlog::warn("starting scan process in {} mode", static_cast<int>(scanType));

            if (scanType == ScanType::EXPRESS)
            {
                // Adjust this part according to your payload format
                std::string payload = "\x00\x00\x00\x00\x00";
                this->_send_payload_cmd(cmd, payload);
            }
            else
            {
                this->_send_cmd(cmd);
            }

            uint8_t dsize;
            bool isSingle;
            uint8_t dtype;

            auto descriptor_result = this->_read_descriptor();
            if (!descriptor_result.has_value())
                return tl::make_unexpected(descriptor_result.error());

            std::tie(dsize, isSingle, dtype) = descriptor_result.value();

            if (dsize != SCAN_TYPE[scanType]["size"])
            {
                return tl::make_unexpected("Wrong get_info reply length");
            }
            if (isSingle)
            {
                return tl::make_unexpected("Not a multiple response mode");
            }
            if (dtype != SCAN_TYPE[scanType]["response"])
            {
                return tl::make_unexpected("Wrong response data type");
            }
            this->scanning = ScanInfo{true, dsize, scanType};
            return nullptr;
        }

        void reset()
        {
            spdlog::info("Resetting the sensor");
            this->_send_cmd(RESET_BYTE);
            std::this_thread::sleep_for(std::chrono::milliseconds(2000));
            this->clean_input();
        }

        std::function<tl::expected<Measure, std::string>()> iter_measures(ScanType scanType =
    ScanType::NORMAL, int maxBufMeas = 3000)
        {
            if (!this->motor_running)
            {
                this->start_motor();
            }

            if (!this->scanning.currently_scanning)
            {
                this->start(scanType);
            }

            // Define a lambda function to generate measures
            auto generator = [this, scanType, maxBufMeas]() -> tl::expected<Measure, std::string>
            {
                while (true)
                {
                    int dsize = scanning.dsize;

                    if (maxBufMeas != 0)
                    {
                        int dataInBuf = this->_serial->available();
                        if (dataInBuf > maxBufMeas)
                        {
                            spdlog::warn(
                                "Too many bytes in the input buffer: {}/{}. \n"
                                "Cleaning buffer...",
                                dataInBuf, maxBufMeas);
                            this->stop();
                            this->start(scanning.type);
                        }
                    }

                    if (scanType == ScanType::NORMAL)
                    {
                        std::vector<uint8_t> raw = this->_read_response(dsize);
                        return _process_scan(raw);
                    }
                    else if (scanType == ScanType::EXPRESS)
                    {
                        if (this->express_trame == 32)
                        {
                            this->express_trame = 0;

                            if (this->express_data == nullptr)
                            {
                                spdlog::debug("reading first time bytes");
                                this->express_data =
   std::make_unique<ExpressPacket>(ExpressPacket(this->_read_response(dsize)));
                            }

                            this->express_old_data = std::move(this->express_data);
                            spdlog::debug("set old_data with start_angle {}", this->express_old_data->start_angle);
                            this->express_data = std::make_unique<ExpressPacket>(ExpressPacket(this->_read_response(dsize)));
                            spdlog::debug("set new_data with start_angle {}", this->express_data->start_angle);
                        }
                        this->express_trame++;
                        spdlog::debug("process scan of frame %d with angle : \n"
                                      "%f and angle new : %f",
                                      this->express_trame,
                                      this->express_old_data->start_angle,
                                      this->express_data->start_angle);
                        Measure measure = _process_express_scan(this->express_old_data, this->express_data->start_angle, this->express_trame);
                        return measure;
                    }
                }
            };

            return generator;
        }

        std::function<std::vector<Measure>()> iter_scans(ScanType scanType = ScanType::NORMAL, int maxBufMeas = 3000, int minLen = 5)
        {
            auto measureIterator = this->iter_measures(scanType, maxBufMeas);

            // Define a lambda function to generate scans
            auto scanGenerator = [measureIterator, minLen]() -> std::vector<Measure>
            {
                std::vector<Measure> scanList;
                do
                {
                    tl::expected<Measure, std::string> maybe_measure = measureIterator();
                    if (!maybe_measure.has_value())
                    {
                        continue;
                    }
                    Measure measure = maybe_measure.value();
                    bool newScan = measure.newScan;
                    if (newScan)
                    {
                        if (scanList.size() > minLen)
                        {
                            break;
                        }
                        scanList.clear();
                    }
                    if (measure.distance > 0)
                    {
                        scanList.push_back(measure);
                    }
                }
                while (true);
                return scanList;
            };
            return scanGenerator;
        }

    private:
        void _set_pwm(int pwm)
        {
            std::string payload;
            payload.push_back(static_cast<uint8_t>(pwm & 0xFF));
            payload.push_back(static_cast<uint8_t>((pwm » 8) & 0xFF));
            this->_send_payload_cmd(SET_PWM_BYTE, payload);
        }

        void _send_payload_cmd(uint8_t cmd, const std::string &payload)
        {
            // Calculate the size
            uint8_t size = static_cast<uint8_t>(payload.size());

            // Construct the request string
            std::string req;
            req += static_cast<uint8_t>(SYNC_BYTE);
            req += static_cast<uint8_t>(cmd);
            req += static_cast<uint8_t>(size);
            req += payload;

            // Calculate the checksum
            uint8_t checksum = 0;
```

```
            for (const uint8_t &c : req)
            {
                checksum ^= static_cast<uint8_t>(c);
            }

            req += static_cast<uint8_t>(checksum);

            this->_serial->write(req);
            spdlog::debug("Command sent: {}", spdlog::to_hex(req));
        }

        void _send_cmd(uint8_t cmd)
        {
            std::string req;
            req += static_cast<uint8_t>(SYNC_BYTE);
            req += static_cast<uint8_t>(cmd);

            this->_serial->write(req);
            spdlog::debug("Command sent: {}", spdlog::to_hex(req));
        }

        tl::expected<
            std::tuple<uint8_t, bool, uint8_t>,
            std::string>
        _read_descriptor()
        {
            // Read descriptor packet
            std::array<uint8_t, DESCRIPTOR_LEN> descriptor;
            memset(descriptor.data(), '\0', sizeof(descriptor));
            this->_serial->read(descriptor.data(), DESCRIPTOR_LEN);
            spdlog::debug("Received descriptor: {}", spdlog::to_hex(descriptor));

            if (descriptor.size() != DESCRIPTOR_LEN)
            {
                return tl::make_unexpected("Descriptor length mismatch");
            }
            else if (descriptor[0] != SYNC_BYTE || descriptor[1] != SYNC_BYTE2)
            {
                return tl::make_unexpected("Incorrect descriptor starting bytes");
            }

            bool isSingle = descriptor[5] == 0;
            return std::make_tuple(descriptor[2], isSingle, descriptor[6]);
        }

        std::vector<uint8_t> _read_response(int dsize)
        {
            spdlog::debug("Trying to read response: {} bytes", dsize);

            std::vector<uint8_t> data;
            data.reserve(dsize);

            while (this->_serial->available() < dsize)
            {
                std::this_thread::sleep_for(std::chrono::milliseconds(1));
            }

            this->_serial->read(data, dsize);

            spdlog::debug("Received data: {}", spdlog::to_hex(data));
            return data;
        }

        std::string convertToHexString(uint8_t value)
        {
            // Convert a uint8_t to a hexadecimal string
            std::stringstream stream;
            stream « std::hex « std::uppercase « std::setw(2) « std::setfill('0') «
    static_cast<int>(value);
            return stream.str();
        }

        tl::expected<Measure, std::string> _process_scan(const std::vector<uint8_t> &raw)
        {
            Measure measurementData;

            bool newScan = static_cast<bool>(raw[0] & 0b1);
            bool inversedNewScan = static_cast<bool>((raw[0] » 1) & 0b1);
            int quality = static_cast<int>(raw[0] » 2);

            if (newScan == inversedNewScan)
            {
                return tl::make_unexpected("New scan flags mismatch");
            }

            int checkBit = static_cast<int>(raw[1] & 0b1);
            if (checkBit != 1)
            {
                return tl::make_unexpected("Check bit not equal to 1");
            }

            int anglePart1 = static_cast<int>(raw[1] » 1);
            int anglePart2 = static_cast<int>(raw[2]) « 7;
            float angle = (anglePart1 + anglePart2) / 64.0;

            int distancePart1 = static_cast<int>(raw[3]);
            int distancePart2 = static_cast<int>(raw[4]) « 8;
            float distance = (distancePart1 + distancePart2) / 4.0;

            measurementData.newScan = newScan;
            measurementData.quality = quality;
            measurementData.angle = angle;
            measurementData.distance = distance;

            return measurementData;
        }

        Measure _process_express_scan(std::unique_ptr<ExpressPacket> &data, float newAngle, int trame)
        {
            Measure measurementData;

            bool newScan = (newAngle < data->start_angle) && (trame == 1);

            float angle = std::fmod((data->start_angle + ((newAngle - data->start_angle) / 32 * trame -
       data->angle[trame - 1])), 360);
            float distance = data->distance[trame - 1];

            measurementData.newScan = newScan;
            measurementData.quality = 0; // Replace this with your actual quality value
            measurementData.angle = angle;
            measurementData.distance = distance;

            return measurementData;
        }

    private:
        std::unique_ptr<serial::Serial> _serial = nullptr;

        std::string port;
        uint32_t baudrate;

        int _motor_speed = DEFAULT_MOTOR_PWM;
        bool motor_running = false;
        ScanInfo scanning = ScanInfo{false, 0, ScanType::NORMAL};
        int express_trame = 32;

        std::unique_ptr<ExpressPacket> express_data = nullptr;
        std::unique_ptr<ExpressPacket> express_old_data = nullptr;
    };
}

#endif
```

8.95 utils/include/utils/TypeName.hpp File Reference

```
#include <array>
#include <cstddef>
```

Classes

- struct utils::RawTypeNameFormat

Namespaces

- namespace utils

Functions

- template<typename T >
  constexpr const auto & utils::RawTypeName ()
- constexpr bool utils::GetRawTypeNameFormat (RawTypeNameFormat ∗format)
- template<typename T >
  constexpr auto utils::CexprTypeName ()
- template<typename T >
  const char ∗ utils::TypeName ()
- template<typename T >
  const char ∗ utils::TypeName (const T &)

8.96 TypeName.hpp

Go to the documentation of this file.

```
// From: https://stackoverflow.com/a/59522794

#include <array>
#include <cstddef>

namespace utils
{
    template <typename T>
    constexpr const auto &RawTypeName()
     {
#ifdef _MSC_VER
         return __FUNCSIG__;
#else
         return __PRETTY_FUNCTION__;
#endif
     }

     struct RawTypeNameFormat
     {
         std::size_t leading_junk = 0, trailing_junk = 0;
     };

     // Returns `false` on failure.
     inline constexpr bool GetRawTypeNameFormat(RawTypeNameFormat *format)
     {
         const auto &str = RawTypeName<int>();
         for (std::size_t i = 0;; i++)
         {
             if (str[i] == 'i' && str[i + 1] == 'n' && str[i + 2] == 't')
             {
                 if (format)
                 {
                     format->leading_junk = i;
                     format->trailing_junk = sizeof(str) - i - 3 - 1; // `3` is the length of "int", `1`
     is the space for the null terminator.
                 }
                 return true;
             }
         }
         return false;
     }

     inline static constexpr RawTypeNameFormat format =
         []
     {
         static_assert(GetRawTypeNameFormat(nullptr), "Unable to figure out how to generate type names on
     this compiler.");
         RawTypeNameFormat format;
         GetRawTypeNameFormat(&format);
         return format;
     }();

     // Returns the type name in a `std::array<char, N>` (null-terminated).
     template <typename T>
     [[nodiscard]] constexpr auto CexprTypeName()
     {
         constexpr std::size_t len = sizeof(RawTypeName<T>()) - format.leading_junk -
     format.trailing_junk;
         std::array<char, len> name{};
         for (std::size_t i = 0; i < len - 1; i++)
             name[i] = RawTypeName<T>()[i + format.leading_junk];
```

```
        return name;
    }

    template <typename T>
    [[nodiscard]] const char *TypeName()
    {
        static constexpr auto name = CexprTypeName<T>();
        return name.data();
    }
    template <typename T>
    [[nodiscard]] const char *TypeName(const T &)
    {
        return TypeName<T>();
    }
}
```

8.97 utils/include/utils/Utility.hpp File Reference

```
#include <string>
#include <sstream>
#include <vector>
```

Namespaces

- namespace utils

8.98 Utility.hpp

Go to the documentation of this file.

```
#ifndef UTILITY_HPP
#define UTILITY_HPP

#pragma once

#include <string>
#include <sstream>
#include <vector>

namespace utils {
    // https://stackoverflow.com/a/46931770
    static std::vector<std::string> split(const std::string& s, const int from_index, const int to_index,
    char delim) {
        std::vector<std::string> result;
        std::stringstream ss(s);
        std::string item;
        int index = 0;

        while (getline(ss, item, delim)) {
            if (index >= from_index && index < to_index)
                result.push_back(item);
            index++;
        }

        return result;
    }

    // https://stackoverflow.com/a/46711735
    static constexpr uint32_t hash(const std::string_view s) noexcept
    {
        uint32_t hash = 5381;

        for (const char* c = s.data(); c < s.data() + s.size(); ++c)
            hash = ((hash « 5) + hash) + (unsigned char)*c;

        return hash;
    }

    // From: https://stackoverflow.com/a/5665377
    // To escape HTML characters
    static void encode(std::string& data) {
        std::string buffer;
        buffer.reserve(data.size());
        for (size_t pos = 0; pos != data.size(); ++pos) {
            switch (data[pos]) {
            case '&':  buffer.append("&amp;");       break;
            case '\"': buffer.append("&quot;");      break;
            case '\'': buffer.append("&apos;");      break;
            case '<':  buffer.append("&lt;");        break;
            case '>':  buffer.append("&gt;");        break;
            default:   buffer.append(&data[pos], 1); break;
            }
        }
        data.swap(buffer);
    }

    // To get class name without namespace
    static std::string getStringAfterLastColon(const std::string& input) {
        size_t last_colon_pos = input.find_last_of("::");

        if (last_colon_pos != std::string::npos) {
            return input.substr(last_colon_pos + 1);
        }
        else {
            return input;
        }
    }
}

#endif
```

Index