Other Code

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 = 15.1515151515151515151515151515151515151515151515

6.2.1 Variable Documentation

6.2.1.1 CM_TO_DISTANCE

```
constexpr double behaviour_tree::global::CM_TO_DISTANCE = 15.1515151515151515151515151515151515151515151515
[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 in-dex)
- 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:

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│
└─────────────────────────────────────────────────┘
```
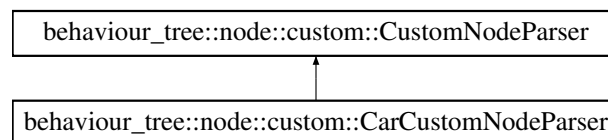
Public Member Functions

- 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:

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:
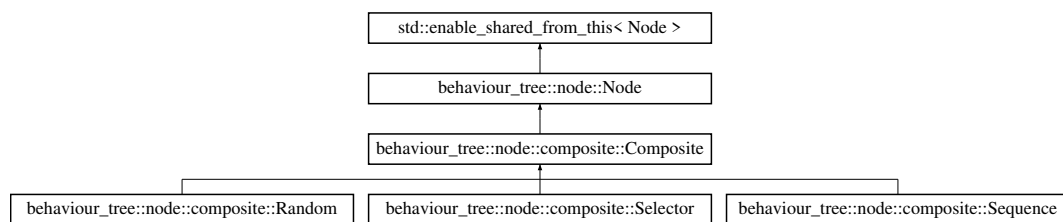
Public Member Functions

- 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:

```
┌────────────────────────────────────────┐
│  std::enable_shared_from_this< Context > │
└────────────────────────────────────────┘
                    ▲
                    │
┌────────────────────────────────────────┐
│        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:

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:
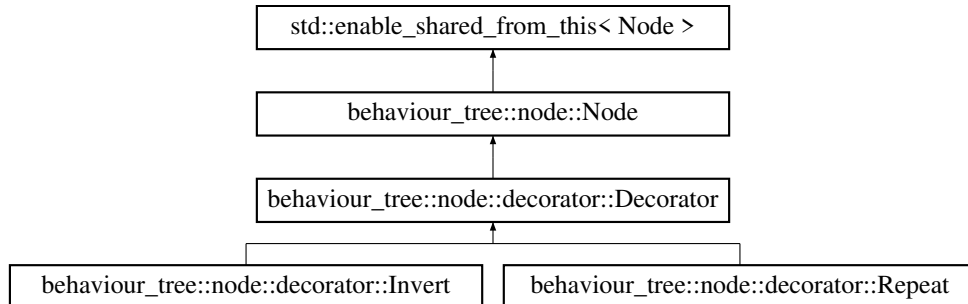
Public Member Functions

- 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.

Generated by Doxygen

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:
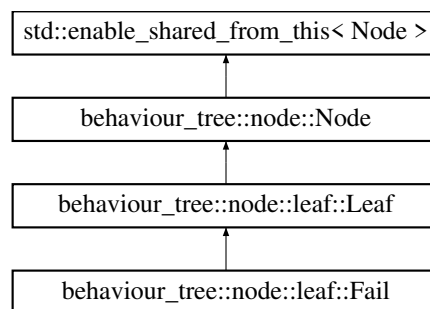
Public Member Functions

- 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│
└─────────────────────────────────────────────────┘
```
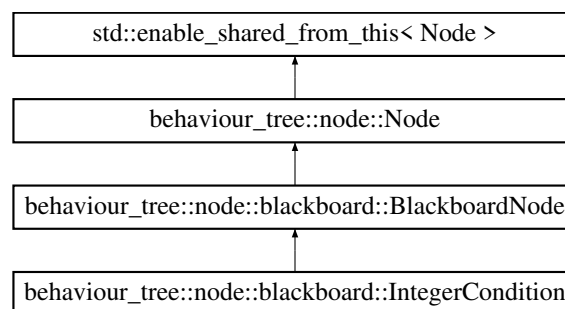
Public Member Functions

- 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:
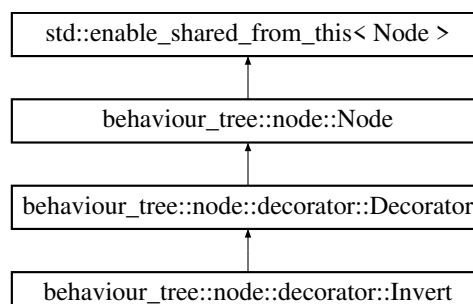
Public Member Functions

- 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:
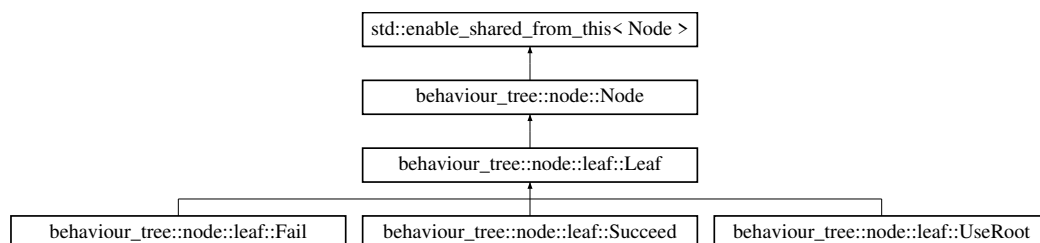
Public Member Functions

- 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:
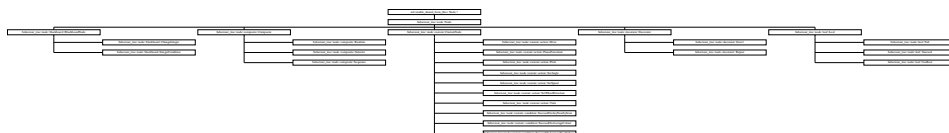
Public Member Functions

- 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:

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:

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:
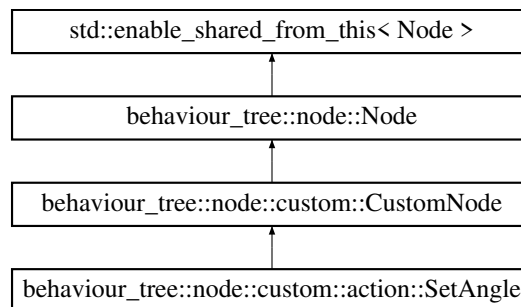
Public Member Functions

- 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:

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:

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:
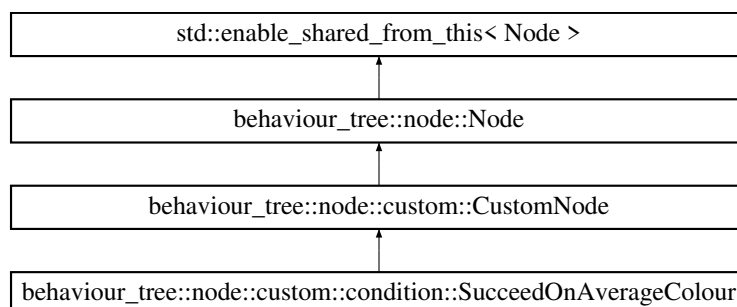
Public Member Functions

- 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  │
└─────────────────────────────────────────────────────────────┘
```
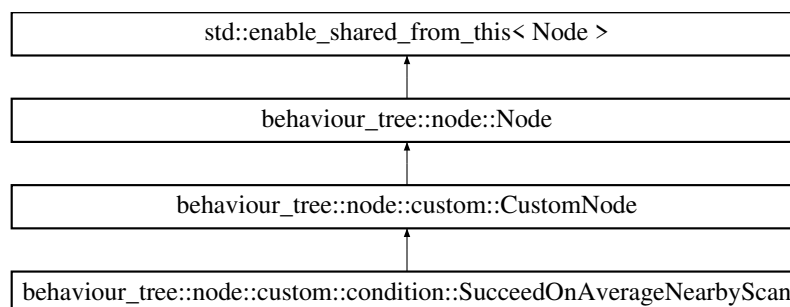
Public Member Functions

- 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│
└─────────────────────────────────────────┘
```
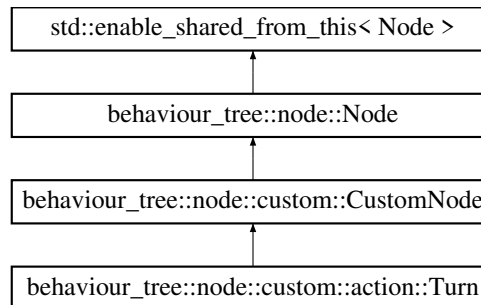
Public Member Functions

- 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   │
└─────────────────────────────────────────┘
```
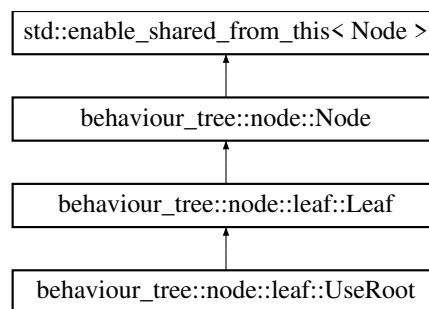
Public Member Functions

- 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.
```cpp
1 #ifndef BEHAVIOUR_TREE_HPP
2 #define BEHAVIOUR_TREE_HPP
3
4 #pragma once
5
6 /*
7 XML format inspired by: https://github.com/telcy/modular-behavior-tree &
       https://github.com/Defaultio/BehaviorTree3
8 */
9
10 #include "Root.hpp"
11
12 namespace behaviour_tree
13 {
14     class BehaviourTree
15     {
16     public:
17         BehaviourTree(unsigned int cycle_limit, std::vector<std::shared_ptr<Root» roots) :
       cycle_limit(cycle_limit), roots(std::move(roots))
18         {
19         }
20
21         void start()
22         {
23             for (auto& root : this->roots)
```

```
24              {
25                  if (root->getId() == "Main")
26                  {
27                      this->root_to_use = root;
28                      return;
29                  }
30              }
31              this->root_to_use = this->roots[0];
32          }
33
34          Status tick(const int tick_count, std::shared_ptr<Context> context)
35          {
36              const Status status = this->root_to_use->tick(tick_count, context);
37              if (status != Status::Running)
38              {
39                  this->cycles++;
40              }
41              return status;
42          }
43
44          Status UseRoot(const int tick_count, std::shared_ptr<Context> context, const std::string& id) {
45              for (auto& root : this->roots)
46              {
47                  if (root->getId() == id)
48                  {
49                      return root->tick(tick_count, context);
50                  }
51              }
52              return Status::Failure;
53          }
54
55          bool canRun() const {
56              if (this->cycle_limit <= 0)
57              {
58                  return true;
59              }
60              return this->cycles < this->cycle_limit;
61          }
62
63          void incrementCycle() {
64              this->cycles++;
65          }
66
67          void resetCycles() {
68              this->cycles = 0;
69          }
70
71          const std::string toString() const {
72              std::string out;
73              for (auto& root : this->roots)
74              {
75                  out += root->toString();
76              }
77              return fmt::format(R"(<BehaviourTree cycle_limit='{}'>{}</BehaviourTree>)",
78     this->cycle_limit, out);
          }
79
80     private:
81          int cycles = 0;
82          std::shared_ptr<Root> root_to_use;
83
84          const std::vector<std::shared_ptr<Root» roots;
85          const unsigned int cycle_limit;
86     };
87 }
88
89 #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.
```
1 #ifndef BEHAVIOUR_TREE_PARSER_HPP
2 #define BEHAVIOUR_TREE_PARSER_HPP
3
4 #pragma once
5
6 #include <filesystem>
7 #include <string>
8 #include <numeric>
9
10 #include <tl/expected.hpp>
11 #include <pugixml.hpp>
12 #include <fmt/format.h>
13
14 #include "utils/Utility.hpp"
15
16 #include "BehaviourTree.hpp"
17 #include "Root.hpp"
18 #include "node/Node.hpp"
19
20 #include "node/composite/Composite.hpp"
21 #include "node/composite/Sequence.hpp"
22 #include "node/composite/Selector.hpp"
23 #include "node/composite/Random.hpp"
24
25 #include "node/decorator/Decorator.hpp"
26 #include "node/decorator/Repeat.hpp"
27 #include "node/decorator/Invert.hpp"
28
29 #include "node/leaf/Leaf.hpp"
30 #include "node/leaf/Succeed.hpp"
31 #include "node/leaf/Fail.hpp"
32 #include "node/leaf/UseRoot.hpp"
```

```
33
34 #include "node/blackboard/ChangeInteger.hpp"
35 #include "node/blackboard/IntegerCondition.hpp"
36
37 #include "node/custom/CustomNode.hpp"
38
39 #include "node/custom/CustomNodeParser.hpp"
40
41 using namespace behaviour_tree::node;
42 using namespace behaviour_tree::node::composite;
43 using namespace behaviour_tree::node::decorator;
44 using namespace behaviour_tree::node::leaf;
45 using namespace behaviour_tree::node::blackboard;
46 using namespace behaviour_tree::node::custom;
47
48 namespace behaviour_tree
49 {
50     static constexpr int STARTING_INDEX = 1;
51
52     class BehaviourTreeParser
53     {
54     public:
55         BehaviourTreeParser(BehaviourTreeParser const&) = delete;
56         void operator=(BehaviourTreeParser const&) = delete;
57
58         static BehaviourTreeParser& instance()
59         {
60             static BehaviourTreeParser instance;
61             return instance;
62         }
63
64         void setCustomNodeParser(std::shared_ptr<CustomNodeParser> custom_node_parser)
65         {
66             this->custom_node_parser = std::move(custom_node_parser);
67         }
68
69         tl::expected<std::shared_ptr<BehaviourTree>, std::string> parseXML(const std::string& xml)
70         {
71             pugi::xml_document doc;
72             pugi::xml_parse_result result = doc.load_string(xml.c_str());
73             if (!result)
74             {
75                 return tl::make_unexpected("XML parse error: " + std::string(result.description()));
76             }
77             return parse(doc);
78         }
79
80         tl::expected<std::shared_ptr<BehaviourTree>, std::string> parseFileXML(const std::string&
81     file_path)
        {
82             if (std::filesystem::exists(file_path) == false)
83             {
84                 return tl::make_unexpected("File [" + file_path + "] does not exist");
85             }
86             pugi::xml_document doc;
87             pugi::xml_parse_result result = doc.load_file(file_path.c_str());
88             if (!result)
89             {
90                 return tl::make_unexpected("XML [" + file_path + "] parsed with errors");
91             }
92             return parse(doc);
93         }
94
95     private:
96         BehaviourTreeParser() {}
97
98         std::shared_ptr<CustomNodeParser> custom_node_parser;
99
100         tl::expected<std::shared_ptr<BehaviourTree>, std::string> parse(pugi::xml_document& doc)
101         {
102             pugi::xml_node behaviour_tree_node = doc.child("BehaviourTree");
103             if (!behaviour_tree_node)
104             {
105                 return tl::make_unexpected(R"(No "BehaviourTree" node found)");
106             }
107             std::vector<std::shared_ptr<Root» roots;
108             for (pugi::xml_node node = behaviour_tree_node.child("Root"); node; node =
109     node.next_sibling("Root"))
            {
110                 const auto maybe_root = parseRoot(node, STARTING_INDEX + roots.size());
111                 if (!maybe_root.has_value())
112                 {
113                     return tl::make_unexpected(maybe_root.error());
114                 }
115                 roots.push_back(std::move(maybe_root.value()));
116             }
117             if (roots.size() <= 0)
```

```
118                 {
119                     return tl::make_unexpected(R"(No "Root" nodes found in BehaviourTree)");
120                 }
121             const int cycle_limit = behaviour_tree_node.attribute("cycle_limit").as_int(0);
122             return std::make_shared<BehaviourTree>(
123                 BehaviourTree(cycle_limit, std::move(roots)));
124         }
125
126     private:
127         tl::expected<std::shared_ptr<Root>, std::string> parseRoot(const pugi::xml_node& node, const int
    index)
128         {
129             int child_count = 0;
130             for (pugi::xml_node child = node.first_child(); child; child = child.next_sibling())
131                 ++child_count;
132             const std::string id = node.attribute("id").as_string();
133             if (child_count == 0)
134             {
135                 return std::make_shared<Root>(
136                     Root(id, nullptr));
137             }
138             if (child_count != 1)
139             {
140                 return tl::make_unexpected(fmt::format(R"(Root node must have only at most one child |
    Root:['{}',{}])", id, index));
141             }
142             const pugi::xml_node child = node.first_child();
143             const auto maybe_child_node = parseChild(child, STARTING_INDEX);
144             if (!maybe_child_node.has_value())
145             {
146                 return tl::make_unexpected(fmt::format(R"({}<-Root:['{}',{}])",
    maybe_child_node.error(), id, index));
147             }
148             return std::make_shared<Root>(
149                 Root(id, std::move(maybe_child_node.value())));
150         }
151
152         tl::expected<std::shared_ptr<node::Node>, std::string> parseChild(const pugi::xml_node& node,
    const int index)
153         {
154             const std::string name_attribute = node.attribute("name").as_string();
155             const std::string& node_name = node.name();
156             switch (utils::hash(node_name))
157             {
158 #pragma region Composite Node
159             case utils::hash("Sequence"):
160             {
161                 return parseComposite(node, index, CompositeType::Sequence);
162             }
163             case utils::hash("Selector"):
164             {
165                 return parseComposite(node, index, CompositeType::Selector);
166             }
167             case utils::hash("Random"):
168             {
169                 return parseComposite(node, index, CompositeType::Random);
170             }
171 #pragma endregion
172 #pragma region Decorator Node
173             case utils::hash("Repeat"):
174             {
175                 return parseDecorator(node, index, DecoratorType::Repeat);
176             }
177             case utils::hash("Invert"):
178             {
179                 return parseDecorator(node, index, DecoratorType::Invert);
180             }
181 #pragma endregion
182 #pragma region Leaf Node
183             case utils::hash("Succeed"):
184             {
185                 return std::make_shared<Succeed>(Succeed(name_attribute));
186             }
187             case utils::hash("Fail"):
188             {
189                 return std::make_shared<Fail>(Fail(name_attribute));
190             }
191             case utils::hash("UseRoot"):
192             {
193                 return std::make_shared<UseRoot>(
194                     UseRoot(
195                         name_attribute,
196                         node.attribute("id").as_string()));
197             }
198 #pragma endregion
199 #pragma region Blackboard Leaf Node
200             case utils::hash("Blackboard:ChangeInteger"):
```

```
201                    {
202                        return parseBlackboardNode(node, index, BlackboardType::ChangeInteger);
203                    }
204                case utils::hash("Blackboard:IntegerCondition"):
205                    {
206                        return parseBlackboardNode(node, index, BlackboardType::IntegerCondition);
207                    }
208 #pragma endregion
209                default:
210                    {
211                        return this->parseCustomNode(node, index);
212                    }
213                    }
214            }
215
216        tl::expected<std::shared_ptr<node::decorator::Decorator>, std::string> parseDecorator(const
    pugi::xml_node& node, const int index, const DecoratorType decorator_type)
217            {
218                const std::string name_attribute = node.attribute("name").as_string();
219                const pugi::xml_node child = node.first_child();
220                if (!child)
221                    {
222                        return tl::make_unexpected(fmt::format(R"(Decorator node must have a child |
    {}:['{}',{}])", std::string(node.name()), name_attribute, index));
223                    }
224                const auto maybe_child_node = parseChild(child, STARTING_INDEX);
225                if (!maybe_child_node.has_value())
226                    {
227                        return tl::make_unexpected(maybe_child_node.error());
228                    }
229                switch (decorator_type)
230                    {
231                case DecoratorType::Invert:
232                    {
233                        return std::make_shared<Invert>(
234                            Invert(
235                                name_attribute,
236                                std::move(maybe_child_node.value())));
237                    }
238                case DecoratorType::Repeat:
239                    {
240                        std::string amount_string = node.attribute("amount").as_string();
241                        // Convert to lower case: https://stackoverflow.com/a/313990
242                        std::transform(amount_string.begin(), amount_string.end(), amount_string.begin(),
243                            [](unsigned char c)
244                            { return std::tolower(c); });
245                        unsigned long count = 0;
246                        if (amount_string == "inf")
247                            {
248                                count = std::numeric_limits<unsigned long>::max();
249                            }
250                        else
251                            {
252                                try
253                                    {
254                                        count = std::stoll(amount_string);
255                                        if (count < 0)
256                                            {
257                                                return tl::make_unexpected(fmt::format(R"(Invalid amount: '{}' |
    Repeat:['{}',{}])", amount_string, name_attribute, index));
258                                            }
259                                    }
260                                catch (const std::exception& _)
261                                    {
262                                        return tl::make_unexpected(fmt::format(R"(Invalid amount: '{}' |
    Repeat:['{}',{}])", amount_string, name_attribute, index));
263                                    }
264                            }
265
266                        return std::make_shared<Repeat>(
267                            Repeat(
268                                name_attribute,
269                                std::move(maybe_child_node.value()),
270                                count,
271                                node.attribute("break_on_fail").as_bool(false)));
272                    }
273                default:
274                        return tl::make_unexpected(fmt::format(R"(Got an invalid Decorator node type |
    {}:['{}',{}])", std::string(node.name()), name_attribute, index));
275                    }
276            }
277
278        tl::expected<std::shared_ptr<node::blackboard::BlackboardNode>, std::string>
    parseBlackboardNode(const pugi::xml_node& node, const int index, const BlackboardType
    blackboard_node_type)
279            {
280                const std::string name_attribute = node.attribute("name").as_string();
```

```
281                const std::string variable_name = node.attribute("variable_name").as_string();
282                if (variable_name.empty()) {
283                    return tl::make_unexpected(fmt::format(R"(variable_name cannot be empty |
       Blackboard:ChangeInteger:['{}',{}])", name_attribute, index));
284                }
285                const int value = node.attribute("value").as_int();
286                switch (blackboard_node_type)
287                {
288                case BlackboardType::ChangeInteger:
289                {
290                    const tl::expected<IntegerChangeType, std::string> maybe_integer_change_type = [&]() {
291                        const std::string integer_change_type_attribute =
       node.attribute("integer_change_type").as_string();
292                        tl::expected<IntegerChangeType, std::string> result;
293                        switch (utils::hash(integer_change_type_attribute))
294                        {
295                        case utils::hash("Set"):
296                        {
297                            result = IntegerChangeType::Set;
298                            break;
299                        }
300                        case utils::hash("Add"):
301                        {
302                            result = IntegerChangeType::Add;
303                            break;
304                        }
305                        case utils::hash("Subtract"):
306                        {
307                            result = IntegerChangeType::Subtract;
308                            break;
309                        }
310                        default:
311                        {
312                            result = tl::make_unexpected(fmt::format(R"(Invalid integer_change_type: '{}' |
       Blackboard:ChangeInteger:['{}',{}])", integer_change_type_attribute, name_attribute, index));
313                            break;
314                        }
315                        };
316                        return result;
317                    }
318                    ();
319                    if (!maybe_integer_change_type.has_value()) {
320                        return tl::make_unexpected(maybe_integer_change_type.error());
321                    }
322                    return std::make_shared<ChangeInteger>(ChangeInteger(
323                        name_attribute,
324                        variable_name,
325                        maybe_integer_change_type.value(),
326                        value
327                    ));
328                }
329                case BlackboardType::IntegerCondition:
330                {
331                    const tl::expected<ConditionOperatorType, std::string> maybe_condition_operator_type =
       [&]() {
332                        const std::string condition_operator_type_attribute =
       node.attribute("condition_operator_type").as_string();
333                        tl::expected<ConditionOperatorType, std::string> result;
334                        switch (utils::hash(condition_operator_type_attribute))
335                        {
336                        case utils::hash("="): {
337                            result = ConditionOperatorType::Equal;
338                            break;
339                        }
340                        case utils::hash(">"): {
341                            result = ConditionOperatorType::GreaterThan;
342                            break;
343                        }
344                        case utils::hash(">="): {
345                            result = ConditionOperatorType::GreaterThanOrEqual;
346                            break;
347                        }
348                        case utils::hash("<"): {
349                            result = ConditionOperatorType::LessThan;
350                            break;
351                        }
352                        case utils::hash("<="): {
353                            result = ConditionOperatorType::LessThanOrEqual;
354                            break;
355                        }
356                        case utils::hash("!="): {
357                            result = ConditionOperatorType::NotEqual;
358                            break;
359                        }
360                        default:
361                        {
362                            result = tl::make_unexpected(fmt::format(R"(Invalid condition_operator_type:
```

```
            '{}' | Blackboard:IntegerCondition:['{}',{}])", condition_operator_type_attribute, name_attribute,
            index));
363                         break;
364                     }
365                     };
366                     return result;
367                 }
368             ();
369             if (!maybe_condition_operator_type.has_value()) {
370                 return tl::make_unexpected(maybe_condition_operator_type.error());
371             }
372             return std::make_shared<IntegerCondition>(IntegerCondition(
373                 name_attribute,
374                 variable_name,
375                 maybe_condition_operator_type.value(),
376                 value
377             ));
378         }
379         default:
380             return tl::make_unexpected(fmt::format(R"(Got an invalid Blackboard node type |
            {}:['{}',{}])", std::string(node.name()), name_attribute, index));
381         }
382     }
383
384     tl::expected<std::shared_ptr<Composite>, std::string> parseComposite(const pugi::xml_node& node,
        const int index, const CompositeType composite_type)
385     {
386         const std::string name_attribute = node.attribute("name").as_string();
387         std::vector<std::shared_ptr<node::Node» children;
388         for (pugi::xml_node child = node.first_child(); child; child = child.next_sibling())
389         {
390             auto maybe_node = parseChild(child, STARTING_INDEX + children.size());
391             if (!maybe_node.has_value())
392             {
393                 return tl::make_unexpected(fmt::format(R"({}<-{}:['{}',{}])", maybe_node.error(),
            std::string(node.name()), name_attribute, index));
394             }
395             children.push_back(std::move(maybe_node.value()));
396         }
397         if (children.size() <= 0)
398         {
399             return tl::make_unexpected(fmt::format(R"(Composite node must have at least one child |
            {}:['{}',{}])", std::string(node.name()), name_attribute, index));
400         }
401         switch (composite_type)
402         {
403         case CompositeType::Sequence:
404             return std::make_shared<Sequence>(Sequence(name_attribute, std::move(children)));
405         case CompositeType::Selector:
406             return std::make_shared<Selector>(Selector(name_attribute, std::move(children)));
407         case CompositeType::Random:
408             return std::make_shared<Random>(Random(name_attribute, std::move(children)));
409         default:
410             return tl::make_unexpected(fmt::format(R"(Got an invalid Composite node type |
            {}:['{}',{}])", std::string(node.name()), name_attribute, index));
411         }
412     }
413
414     tl::expected<std::shared_ptr<node::custom::CustomNode>, std::string> parseCustomNode(const
        pugi::xml_node& node, const int index)
415     {
416         return this->custom_node_parser->parseCustomNode(node, index);
417     }
418     };
419 }
420
421 #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.
```
1 #ifndef BEHAVIOUR_TREE_CONTEXT_HPP
2 #define BEHAVIOUR_TREE_CONTEXT_HPP
3
4 #pragma once
5
6 #include <algorithm>
7 #include <memory>
8 #include <string>
9 #include <vector>
10 #include <variant>
11 #include <unordered_map>
12
13 #include "Status.hpp"
14
15 namespace behaviour_tree
16 {
17     class BehaviourTree;
18     namespace node
19     {
20         class Node;
21         namespace composite
22         {
23             class Composite;
24         }
25     }
26 }
27
28 namespace behaviour_tree
29 {
30     class Context : public std::enable_shared_from_this<Context>
31     {
32     public:
33         Context(std::shared_ptr<BehaviourTree> behaviour_tree);
34         void update(const int tick_count);
35         Status UseRoot(const int tick_count, const std::string& id);
36
37         std::shared_ptr<BehaviourTree> getBehaviourTree() const { return this->behaviour_tree; };
38
39         void pushNodeTrace(std::pair<std::shared_ptr<node::Node>, int> node_trace);
40         void popNode();
41
42         bool canRun() const;
```

```
43
44        void setBlackboard(const std::string& key, const int value);
45        int getBlackboard(const std::string& key) const;
46        int containsBlackboard(const std::string& key) const;
47
48        // Necessary for the class to be a polymorphic
49        virtual void _() {};
50    private:
51        std::shared_ptr<BehaviourTree> behaviour_tree;
52        std::vector<std::pair<std::shared_ptr<node::Node>, int» node_trace_list;
53
54        std::unordered_map<std::string, int> blackboard;
55    };
56 }
57
58 #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.
```
1 #ifndef BEHAVIOUR_TREE_BLACKBOARD_NODE_HPP
2 #define BEHAVIOUR_TREE_BLACKBOARD_NODE_HPP
3
4 #pragma once
5
6 #include "../Node.hpp"
7
8 #include "BlackboardType.hpp"
9
10 namespace behaviour_tree::node::blackboard
11 {
12     class BlackboardNode : public Node
13     {
14     public:
15         BlackboardNode(const std::string& name) : Node(name)
16         {
17         }
18
19         virtual const BlackboardType type() const = 0;
20     };
21 }
22
23 #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.
```
1 #ifndef BEHAVIOUR_TREE_BLACKBOARD_TYPE_HPP
2 #define BEHAVIOUR_TREE_BLACKBOARD_TYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::blackboard
7 {
8     enum class BlackboardType
9     {
10         ChangeInteger,
11         IntegerCondition,
12     };
13 }
14
15 #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.
```cpp
1 #ifndef BEHAVIOUR_TREE_CHANGE_INTEGER_HPP
2 #define BEHAVIOUR_TREE_CHANGE_INTEGER_HPP
3
4 #pragma once
5
6 #include "BlackboardNode.hpp"
7
8 #include "enum/IntegerChangeType.hpp"
9
10 namespace behaviour_tree::node::blackboard
11 {
12     class ChangeInteger final : public BlackboardNode
13     {
14     public:
15         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) {}
16
17         const BlackboardType type() const final override { return BlackboardType::ChangeInteger; }
18
19         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
20         {
21             if (!context->containsBlackboard(this->variable_name)) {
22                 context->setBlackboard(this->variable_name, 0);
23             }
24             int value = context->getBlackboard(this->variable_name);
25             switch (this->integer_change_type) {
26             case IntegerChangeType::Set: {
27                 context->setBlackboard(this->variable_name, this->value);
28                 break;
29             }
30             case IntegerChangeType::Add: {
31                 context->setBlackboard(this->variable_name, value + this->value);
32                 break;
33             }
34             case IntegerChangeType::Subtract: {
35                 context->setBlackboard(this->variable_name, value - this->value);
36                 break;
37             }
38             }
39             return Status::Success;
40         }
41
42         const std::string toString() const final override
43         {
44             const std::string& name = this->getName();
45             const std::string integer_change_type = [&]() {
46                 switch (this->integer_change_type)
47                 {
48                 case IntegerChangeType::Set:
49                     return "Set";
50                 case IntegerChangeType::Add:
51                     return "Add";
52                 case IntegerChangeType::Subtract:
53                     return "Subtract";
54                 default:
55                     return "Invalid";
56                 }
57             }
58             ();
59             if (name != "")
60                 return fmt::format(R"(<Blackboard:ChangeInteger name='{}' variable_name='{}'
     integer_change_type='{}' value='{}'/>)", name, this->variable_name, integer_change_type,
     this->value);
61             else
62                 return fmt::format(R"(<Blackboard:ChangeInteger variable_name='{}'
     integer_change_type='{}' value='{}'/>)", this->variable_name, integer_change_type, this->value);
63         }
64
65     private:
66         const std::string variable_name;
67         const IntegerChangeType integer_change_type;
68         const int value;
69     };
70 }
71
72 #endif
```

8.14 behaviour_tree/include/behaviour_tree/node/blackboard/enum/$\leftarrow$ 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.
```
1 #ifndef BEHAVIOUR_TREE_CONDITION_OPERATOR_TYPE_HPP
2 #define BEHAVIOUR_TREE_CONDITION_OPERATOR_TYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::blackboard
7 {
8     enum class ConditionOperatorType
9     {
10         Equal,
11         NotEqual,
12         GreaterThan,
13         GreaterThanOrEqual,
14         LessThan,
15         LessThanOrEqual,
16     };
17 }
18
19 #endif
```

8.16 behaviour_tree/include/behaviour_tree/node/blackboard/enum/$\leftarrow$ 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.
```
1 #ifndef BEHAVIOUR_TREE_INTEGER_CHANGE_TYPE_HPP
2 #define BEHAVIOUR_TREE_INTEGER_CHANGE_TYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::blackboard
7 {
8     enum class IntegerChangeType
9     {
10         Set,
11         Add,
12         Subtract,
13     };
14 }
15
16 #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.
```
1 #ifndef BEHAVIOUR_TREE_INTEGER_CONDITION_HPP
2 #define BEHAVIOUR_TREE_INTEGER_CONDITION_HPP
3
4 #pragma once
5
6 #include "BlackboardNode.hpp"
7
8 #include "enum/ConditionOperatorType.hpp"
9
10 namespace behaviour_tree::node::blackboard
11 {
12     class IntegerCondition final : public BlackboardNode
13     {
14     public:
15         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) {}
16
17         const BlackboardType type() const final override { return BlackboardType::IntegerCondition; }
18
19         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
```

```
20          {
21              if (!context->containsBlackboard(this->variable_name)) {
22                  context->setBlackboard(this->variable_name, 0);
23              }
24              switch (this->condition_operator)
25              {
26              case ConditionOperatorType::Equal:
27                  return context->getBlackboard(this->variable_name) == this->value ? Status::Success :
    Status::Failure;
28              case ConditionOperatorType::GreaterThan:
29                  return context->getBlackboard(this->variable_name) > this->value ? Status::Success :
    Status::Failure;
30              case ConditionOperatorType::GreaterThanOrEqual:
31                  return context->getBlackboard(this->variable_name) >= this->value ? Status::Success :
    Status::Failure;
32              case ConditionOperatorType::LessThan:
33                  return context->getBlackboard(this->variable_name) < this->value ? Status::Success :
    Status::Failure;
34              case ConditionOperatorType::LessThanOrEqual:
35                  return context->getBlackboard(this->variable_name) <= this->value ? Status::Success :
    Status::Failure;
36              case ConditionOperatorType::NotEqual:
37                  return context->getBlackboard(this->variable_name) != this->value ? Status::Success :
    Status::Failure;
38              default:
39                  return Status::Failure;
40              }
41          }
42
43          const std::string toString() const final override
44          {
45              const std::string& name = this->getName();
46              const std::string condition_operator = [&]() {
47                  switch (this->condition_operator)
48                  {
49                  case ConditionOperatorType::Equal:
50                      return "=";
51                  case ConditionOperatorType::GreaterThan:
52                      return ">";
53                  case ConditionOperatorType::GreaterThanOrEqual:
54                      return ">=";
55                  case ConditionOperatorType::LessThan:
56                      return "<";
57                  case ConditionOperatorType::LessThanOrEqual:
58                      return "<=";
59                  case ConditionOperatorType::NotEqual:
60                      return "!=";
61                  default:
62                      return "Invalid";
63                  }
64              }
65              ();
66              if (name != "")
67                  return fmt::format(R"(<Blackboard:IntegerCondition name='{}' variable_name='{}'
    condition_operator='{}' value='{}'/>)", name, this->variable_name, condition_operator, this->value);
68              else
69                  return fmt::format(R"(<Blackboard:IntegerCondition variable_name='{}'
    condition_operator='{}' value='{}'/>)", this->variable_name, condition_operator, this->value);
70          }
71
72      private:
73          const std::string variable_name;
74          const ConditionOperatorType condition_operator;
75          const int value;
76      };
77 }
78
79 #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.

```
1 #ifndef BEHAVIOUR_TREE_COMPOSITE_HPP
2 #define BEHAVIOUR_TREE_COMPOSITE_HPP
3
4 #pragma once
5
6 #include <vector>
7
8 #include "../Node.hpp"
9 #include "CompositeType.hpp"
10
11 namespace behaviour_tree::node::composite
12 {
13     class Composite : public Node
14     {
15     public:
16         Composite(const std::string& name, std::vector<std::shared_ptr<Node>> children) : Node(name),
    children(std::move(children))
17         {
18         }
19
20         Status tick(const int tick_count, std::shared_ptr<Context> context, const int start_index)
21         {
22             assert(context != nullptr);
23
24             if (!this->started) {
25                 this->start(context);
26                 this->started = true;
27             }
28             Status status = this->run(tick_count, context, start_index);
29             if (status == Status::Success || status == Status::Failure) {
30                 this->finish(context);
31                 this->started = false;
32             }
33             return status;
34         }
35
36         virtual const Status run(const int tick_count, std::shared_ptr<Context> context, const int
    start_index) = 0;
37         virtual const CompositeType type() const = 0;
38
39     protected:
40         std::vector<std::shared_ptr<Node>> children;
41     };
42 }
43
44 #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.](#)
```cpp
1 #ifndef BEHAVIOUR_TREE_COMPOSITETYPE_HPP
2 #define BEHAVIOUR_TREE_COMPOSITETYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::composite
7 {
8     enum class CompositeType
9     {
10         Sequence,
11         Selector,
12         Random,
13     };
14 }
15
16 #endif
```

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

```cpp
#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.
```
1 #ifndef BEHAVIOUR_TREE_RANDOM_HPP
2 #define BEHAVIOUR_TREE_RANDOM_HPP
3
4 #pragma once
5
6 #include <effolkronium/random.hpp>
7
8 #include "Composite.hpp"
9
10 using effolkronium_Random = effolkronium::random_static;
11
12 namespace behaviour_tree::node::composite
13 {
14     class Random final : public Composite
15     {
16     public:
17         Random(const std::string& name, std::vector<std::shared_ptr<Node» children) : Composite(name,
    std::move(children)) {}
18
19         const CompositeType type() const final override { return CompositeType::Random; }
20
21         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
22         {
23             return this->run(tick_count, context, 0);
24         }
25
26         const Status run(const int tick_count, std::shared_ptr<Context> context, const int start_index)
    final override
27         {
28             // Random can never be empty from the BehaviourTreeParser
29             return this->children[effolkronium_Random::get(0, static_cast<int>(this->children.size() -
    1))]->tick(tick_count, context);
30         }
31
32         const std::string toString() const final override {
33             std::string out;
34             for (auto& child : this->children)
35             {
36                 out += child->toString();
37             }
38             const std::string& name = this->getName();
39             if (name != "")
40                 return fmt::format(R"(<Random name='{}'>{}</Random>)", name, out);
41             else
42                 return fmt::format(R"(<Random>{}</Random>)", out);
43         }
44     };
45 }
46
47 #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.

```cpp
1 #ifndef BEHAVIOUR_TREE_SELECTOR_HPP
2 #define BEHAVIOUR_TREE_SELECTOR_HPP
3
4 #pragma once
5
6 #include "Composite.hpp"
7
8 namespace behaviour_tree::node::composite
9 {
10     class Selector final : public Composite
11     {
12     public:
13         Selector(const std::string& name, std::vector<std::shared_ptr<Node» children) : Composite(name,
     std::move(children)) {}
14
15         const CompositeType type() const final override { return CompositeType::Selector; }
16
17         void start(std::shared_ptr<Context> context) final override {
18             this->previous_start_index = -1;
19         }
20
21         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
22         {
23             return this->run(tick_count, context, 0);
24         }
25
26         const Status run(const int tick_count, std::shared_ptr<Context> context, const int start_index)
     final override
27         {
28             for (int i = start_index; i < this->children.size(); i++)
29             {
30                 if (this->previous_start_index != i) {
31                     context->pushNodeTrace(std::make_pair(shared_from_this(), i));
32                 }
33                 auto& child = this->children[i];
34                 auto status = child->tick(tick_count, context);
35                 switch (status) {
36                 case Status::Running:
37                     this->previous_start_index = i;
38                     return Status::Running;
39                 case Status::Failure:
40                     context->popNode();
41                     continue;
42                 case Status::Success:
43                     context->popNode();
44                     return Status::Success;
45                 }
46             }
47             return Status::Failure;
48         }
49
50         const std::string toString() const final override {
51             std::string out;
52             for (auto& child : this->children)
53             {
54                 out += child->toString();
55             }
56             const std::string& name = this->getName();
57             if (name != "")
58                 return fmt::format(R"(<Selector name='{}'>{}</Selector>)", name, out);
59             else
60                 return fmt::format(R"(<Selector>{}</Selector>)", out);
61         }
62
63     private:
64         int previous_start_index = -1;
65     };
66 }
67
68 #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.
```cpp
1 #ifndef BEHAVIOUR_TREE_SEQUENCE_HPP
2 #define BEHAVIOUR_TREE_SEQUENCE_HPP
3
4 #pragma once
5
6 #include "Composite.hpp"
7
8 namespace behaviour_tree::node::composite
9 {
10     class Sequence final : public Composite
11     {
12     public:
13         Sequence(const std::string& name, std::vector<std::shared_ptr<Node» children) : Composite(name,
    std::move(children)) {}
14
15         const CompositeType type() const final override { return CompositeType::Sequence; }
16
17         void start(std::shared_ptr<Context> context) final override {
18             this->previous_start_index = -1;
19         }
20
21         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
22         {
23             return this->run(tick_count, context, 0);
24         }
25
26         const Status run(const int tick_count, std::shared_ptr<Context> context, const int start_index)
    final override
27         {
28             for (int i = start_index; i < this->children.size(); i++)
29             {
30                 if (this->previous_start_index != i) {
31                     context->pushNodeTrace(std::make_pair(shared_from_this(), i));
32                 }
33                 auto& child = this->children[i];
34                 auto status = child->tick(tick_count, context);
35                 switch (status) {
36                 case Status::Running:
37                     this->previous_start_index = i;
38                     return Status::Running;
39                 case Status::Success:
40                     context->popNode();
41                     continue;
42                 case Status::Failure:
43                     context->popNode();
44                     return Status::Failure;
45                 }
46             }
47             return Status::Success;
```

```
48          }
49
50          const std::string toString() const final override {
51              std::string out;
52              for (auto& child : this->children)
53              {
54                  out += child->toString();
55              }
56              const std::string& name = this->getName();
57              if (name != "")
58                  return fmt::format(R"(<Sequence name='{}'>{}</Sequence>)", name, out);
59              else
60                  return fmt::format(R"(<Sequence>{}</Sequence>)", out);
61          }
62
63      private:
64          int previous_start_index = -1;
65      };
66 }
67
68 #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.
```
1 #ifndef BEHAVIOUR_TREE_CUSTOM_NODE_HPP
2 #define BEHAVIOUR_TREE_CUSTOM_NODE_HPP
3
4 #pragma once
5
6 #include "../Node.hpp"
7
8 namespace behaviour_tree::node::custom
9 {
10      class CustomNode : public Node
11      {
12      public:
13          CustomNode(const std::string& name) : Node(name) {
14
15          }
16      };
17 }
18
19 #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.
```
1 #ifndef BEHAVIOUR_TREE_CUSTOM_NODE_PARSER_HPP
2 #define BEHAVIOUR_TREE_CUSTOM_NODE_PARSER_HPP
3
4 #pragma once
5
6 #include <string>
7 #include <memory>
8
9 #include <pugixml.hpp>
10 #include <tl/expected.hpp>
11 #include <fmt/format.h>
12
13 #include "CustomNode.hpp"
14
15 namespace behaviour_tree::node::custom
16 {
17     class CustomNodeParser
18     {
19     public:
20         virtual tl::expected<std::shared_ptr<CustomNode>, std::string> parseCustomNode(const
    pugi::xml_node& node, const int index) = 0;
21     };
22 }
23
24 #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.
```
1 #ifndef BEHAVIOUR_TREE_DECORATOR_HPP
2 #define BEHAVIOUR_TREE_DECORATOR_HPP
3
4 #pragma once
5
6 #include "../Node.hpp"
7
8 #include "DecoratorType.hpp"
9
10 namespace behaviour_tree::node::decorator
11 {
12     class Decorator : public Node
13     {
14     public:
15         Decorator(const std::string& name, std::shared_ptr<Node> child) : Node(name),
    child(std::move(child))
16         {
17         }
18
19         virtual const DecoratorType type() const = 0;
20
21     protected:
22         std::shared_ptr<Node> child;
23     };
24 }
25
26 #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

```
1  #ifndef BEHAVIOUR_TREE_DECORATORTYPE_HPP
2  #define BEHAVIOUR_TREE_DECORATORTYPE_HPP
3
4  #pragma once
5
6  namespace behaviour_tree::node::decorator
7  {
8      enum class DecoratorType
9      {
10         Invert,
11         Repeat,
12     };
13 }
14
15 #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

```
1  #ifndef BEHAVIOUR_TREE_INVERT_HPP
2  #define BEHAVIOUR_TREE_INVERT_HPP
3
4  #pragma once
5
6  #include "Decorator.hpp"
7
8  namespace behaviour_tree::node::decorator
9  {
10     class Invert final : public Decorator
11     {
12     public:
13         Invert(const std::string& name, std::shared_ptr<Node> child) : Decorator(name, std::move(child))
       {}
14
15         const DecoratorType type() const final override { return DecoratorType::Invert; }
16
17         void start(std::shared_ptr<Context> context) final override {
18             context->pushNodeTrace(std::make_pair(shared_from_this(), 0));
19         }
20
21         void finish(std::shared_ptr<Context> context) final override {
22             context->popNode();
23         }
```

```
24
25          const Status run(const int tick_count, std::shared_ptr<Context> context) final override
26          {
27              auto status = this->child->tick(tick_count, context);
28              switch (status)
29              {
30              case Status::Success:
31                  return Status::Failure;
32              case Status::Failure:
33                  return Status::Success;
34              case Status::Running:
35                  return Status::Running;
36              }
37          }
38
39          const std::string toString() const final override {
40              const std::string& name = this->getName();
41              if (name != "")
42                  return fmt::format(R"(<Invert name='{}'>{}</Invert>)", name, this->child->toString());
43              else
44                  return fmt::format(R"(<Invert>{}</Invert>)", this->child->toString());
45          }
46      };
47 }
48
49 #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.
```
1 #ifndef BEHAVIOUR_TREE_REPEAT_HPP
2 #define BEHAVIOUR_TREE_REPEAT_HPP
3
4 #pragma once
5
6 #include <numeric>
7
8 #include "Decorator.hpp"
9
10 namespace behaviour_tree::node::decorator
11 {
12      class Repeat final : public Decorator
13      {
14      public:
15          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)
16          {
```

```
17              }
18
19          const DecoratorType type() const final override { return DecoratorType::Repeat; }
20
21          void start(std::shared_ptr<Context> context) final override {
22              this->count = 0;
23          }
24
25          const Status run(const int tick_count, std::shared_ptr<Context> context) final override
26          {
27              auto status = this->child->tick(tick_count, context);
28              switch (status)
29              {
30              case Status::Success:
31                  this->count++;
32                  if (this->count >= this->amount)
33                  {
34                      return Status::Success;
35                  }
36                  return Status::Running;
37              case Status::Running:
38                  return Status::Running;
39              case Status::Failure:
40                  if (this->break_on_fail)
41                      return Status::Failure;
42                  return Status::Running;
43              }
44          }
45
46          const std::string toString() const final override {
47              const std::string& name = this->getName();
48              if (name != "")
49                  return fmt::format(R"(<Repeat name='{}' amount='{}' break_on_fail='{}'>{}</Repeat>)",
      name, this->getAmount(), this->getBreakOnFail() ? "true" : "false", this->child->toString());
50              else
51                  return fmt::format(R"(<Repeat amount='{}' break_on_fail='{}'>{}</Repeat>)",
      this->getAmount(), this->getBreakOnFail() ? "true" : "false", this->child->toString());
52          }
53
54          const unsigned long getAmount() const {
55              return this->amount;
56          }
57
58          const bool getBreakOnFail() const {
59              return this->break_on_fail;
60          }
61
62          const unsigned long getCount() const {
63              return this->count;
64          }
65
66      private:
67          const unsigned long amount;
68          const bool break_on_fail;
69
70          unsigned long count = 0;
71      };
72 }
73
74 #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.

```cpp
1 #ifndef BEHAVIOUR_TREE_FAIL_HPP
2 #define BEHAVIOUR_TREE_FAIL_HPP
3
4 #pragma once
5
6 #include "Leaf.hpp"
7
8 namespace behaviour_tree::node::leaf
9 {
10     class Fail final : public Leaf
11     {
12     public:
13         Fail(const std::string& name) : Leaf(name) {}
14
15         const LeafType type() const final override { return LeafType::Fail; }
16
17         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
18         {
19             return Status::Failure;
20         }
21
22         const std::string toString() const final override {
23             const std::string& name = this->getName();
24             if (name != "")
25                 return fmt::format(R"(<Fail name='{}'/>)", name);
26             else
27                 return fmt::format(R"(<Fail/>)");
28         }
29     };
30 }
31
32 #endif
```

8.44   behaviour_tree/include/behaviour_tree/node/leaf/Leaf.hpp File Reference

```cpp
#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.
```cpp
1 #ifndef BEHAVIOUR_TREE_LEAF_HPP
2 #define BEHAVIOUR_TREE_LEAF_HPP
3
4 #pragma once
5
6 #include "../Node.hpp"
7
8 #include "LeafType.hpp"
9
10 namespace behaviour_tree::node::leaf
11 {
12     class Leaf : public Node
13     {
14     public:
15         Leaf(const std::string& name) : Node(name)
16         {
17         }
18
19         virtual const LeafType type() const = 0;
20     };
21 }
22
23 #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.
```cpp
1 #ifndef BEHAVIOUR_TREE_LEAFTYPE_HPP
2 #define BEHAVIOUR_TREE_LEAFTYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::leaf
7 {
8     enum class LeafType
9     {
10         Succeed,
11         Fail,
12         UseRoot,
13     };
14 }
15
16 #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.
```cpp
1  #ifndef BEHAVIOUR_TREE_SUCCEED_HPP
2  #define BEHAVIOUR_TREE_SUCCEED_HPP
3
4  #pragma once
5
6  #include "Leaf.hpp"
7
8  namespace behaviour_tree::node::leaf
9  {
10     class Succeed final : public Leaf
11     {
12     public:
13         Succeed(const std::string& name) : Leaf(name) {}
14
15         const LeafType type() const final override { return LeafType::Succeed; }
16
17         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
18         {
19             return Status::Success;
20         }
21
22         const std::string toString() const final override {
23             const std::string& name = this->getName();
24             if (name != "")
25                 return fmt::format(R"(<Succeed name='{}'/>)", name);
26             else
27                 return fmt::format(R"(<Succeed/>)");
28         }
29     };
30 }
31
32 #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.
```
1 #ifndef BEHAVIOUR_TREE_UseRoot_HPP
2 #define BEHAVIOUR_TREE_UseRoot_HPP
3
4 #pragma once
5
6 #include "Leaf.hpp"
7
8 namespace behaviour_tree::node::leaf
9 {
10     class UseRoot final : public Leaf
11     {
12     public:
13         UseRoot(const std::string& name, const std::string& id) : Leaf(name), id(id) {}
14
15         const LeafType type() const final override { return LeafType::UseRoot; }
16
17         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
18         {
19             return context->UseRoot(tick_count, this->id);
20         }
21
22         const std::string& getId() const { return this->id; }
23
24         const std::string toString() const final override {
25             const std::string& name = this->getName();
26             if (name != "")
27                 return fmt::format(R"(<UseRoot name='{}' id='{}'/>)", name, this->getId());
28             else
29                 return fmt::format(R"(<UseRoot id='{}'/>)", this->getId());
30         }
31
32     private:
33         const std::string id;
34     };
35 }
36
37 #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.

```
1 #ifndef BEHAVIOUR_TREE_NODE_HPP
2 #define BEHAVIOUR_TREE_NODE_HPP
3
4 #pragma once
5
6 #include <cassert>
7
8 #include <string>
9 #include <memory>
10
11 #include <fmt/format.h>
12
13 #include "../Status.hpp"
14 #include "../Context.h"
15
16 namespace behaviour_tree::node
17 {
18     class Node : public std::enable_shared_from_this<Node>
19     {
20     public:
21         Node(const std::string& name) : name(name)
22         {
23         }
24
25         virtual void start(std::shared_ptr<Context> context) {}
26         virtual const Status run(const int tick_count, std::shared_ptr<Context> context) = 0;
27         virtual void finish(std::shared_ptr<Context> context) {}
28
29         Status tick(const int tick_count, std::shared_ptr<Context> context)
30         {
31             assert(context != nullptr);
32
33             if (!this->started) {
34                 this->start(context);
35                 this->started = true;
36             }
37             Status status = this->run(tick_count, context);
38             if (status == Status::Success || status == Status::Failure) {
39                 this->finish(context);
40                 this->started = false;
41             }
42             return status;
43         }
44
45         virtual const std::string toString() const = 0;
46
47         const std::string& getName() const
48         {
49             return this->name;
50         }
51
52     protected:
53         const std::string name;
54
55         bool started = false;
56     };
57 }
58
59 #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.
```cpp
1  #ifndef BEHAVIOUR_TREE_ROOT_HPP
2  #define BEHAVIOUR_TREE_ROOT_HPP
3
4  #pragma once
5
6  #include <string>
7  #include <fmt/format.h>
8
9  #include "node/Node.hpp"
10
11 using namespace behaviour_tree::node;
12
13 namespace behaviour_tree
14 {
15     class Root
16     {
17     public:
18         Root(std::string id, std::shared_ptr<node::Node> child) : id(id), child(std::move(child))
19         {
20         }
21
22         const Status tick(const int tick_count, std::shared_ptr<Context>& context)
23         {
24             if (this->child == nullptr) return Status::Success;
25             return this->child->tick(tick_count, context);
26         }
27
28         const std::string toString() const {
29             const std::string& id = this->getId();
30             std::string child_string = this->child != nullptr ? this->child->toString() : "";
31             if (id != "")
32                 return fmt::format(R"(<Root id='{}'>{}</Root>)", id, child_string);
33             else
34                 return fmt::format(R"(<Root>{}</Root>)", child_string);
35         }
36
37         const std::string& getId() const
38         {
39             return this->id;
40         }
41
42     private:
43         std::string id;
44         std::shared_ptr<node::Node> child;
45     };
46 }
47
48 #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.
```
1 #ifndef BEHAVIOUR_TREE_STATUS_HPP
2 #define BEHAVIOUR_TREE_STATUS_HPP
3
4 #pragma once
5
6 namespace behaviour_tree
7 {
8     enum class Status
9     {
10         Success,
11         Failure,
12         Running
13     };
14 }
15
16 #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 = 15.1515151515151515151515151515151515151515151515151515

8.59 distance.h

Go to the documentation of this file.
```
1 #ifndef BEHAVIOUR_TREE_GLOBAL_DISTANCE_H
2 #define BEHAVIOUR_TREE_GLOBAL_DISTANCE_H
3
4 #pragma once
5
6 namespace behaviour_tree::global {
7     constexpr double CM_TO_DISTANCE = 15.15151515151515151515151515151515151515151515151515;
8 }
9
10 #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.
```cpp
1  #ifndef BEHAVIOUR_TREE_DRIVE_HPP
2  #define BEHAVIOUR_TREE_DRIVE_HPP
3
4  #pragma once
5
6  #include <fmt/format.h>
7
8  #include "behaviour_tree/node/custom/CustomNode.hpp"
9
10 #include "behaviour_tree/Context.h"
11 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
12 #include "behaviour_tree/CarContext.hpp"
13 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
14
15 #include "enum/DirectionType.hpp"
16
17 namespace behaviour_tree::node::custom::action
18 {
19     class Drive final : public CustomNode
20     {
21     public:
22         Drive(const std::string &name, const int speed, const DirectionType direction_type) :
       CustomNode(name), speed(speed), direction_type(direction_type)
23         {
24         }
25
26         const static tl::expected<std::shared_ptr<Drive>, std::string> parse(const pugi::xml_node &node,
       const int index, const std::string &name_attribute)
27         {
28             const int speed = node.attribute("speed").as_int();
29             if (speed < 0 || speed > 100)
30             {
31                 return tl::unexpected(fmt::format(R"(Invalid speed: '{}' | Action:Drive:['{}',{}])",
       speed, name_attribute, index));
32             }
33             const tl::expected<DirectionType, std::string> maybe_direction_type =
34                 [&]()
35             {
36                 const std::string direction_type_attribute =
       node.attribute("direction_type").as_string();
37                 tl::expected<DirectionType, std::string> result;
```

```
38                   switch (utils::hash(direction_type_attribute))
39                   {
40                   case utils::hash("Forward"):
41                   {
42                       result = DirectionType::Forward;
43                       break;
44                   }
45                   case utils::hash("Backward"):
46                   {
47                       result = DirectionType::Backward;
48                       break;
49                   }
50                   default:
51                   {
52                       result = tl::make_unexpected(fmt::format(R"(Invalid direction_type: '{}' |
      Action:Drive:['{}',{}])", direction_type_attribute, name_attribute, index));
53                       break;
54                   }
55                   };
56                   return result;
57               }();
58               if (!maybe_direction_type.has_value())
59               {
60                   return tl::make_unexpected(maybe_direction_type.error());
61               }
62               return std::make_shared<Drive>(
63                   Drive(
64                       name_attribute,
65                       speed,
66                       maybe_direction_type.value()));
67           }
68
69           const Status run(const int tick_count, std::shared_ptr<Context> context) final override
70           {
71 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
72               std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
73               auto car_system = car_context->getCarSystem();
74               const MovementSystem *movement_system = car_system->getMovementSystem();
75               switch (this->getDirectionType())
76               {
77               case DirectionType::Backward:
78               {
79                   movement_system->setRearWheelsDirectionToBackward();
80                   break;
81               }
82               case DirectionType::Forward:
83               {
84                   movement_system->setRearWheelsDirectionToForward();
85                   break;
86               }
87               }
88               movement_system->setRearWheelsSpeed(this->getSpeed());
89 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
90               return Status::Success;
91           }
92
93           const std::string toString() const final override
94           {
95               const std::string &name = this->getName();
96               std::string direction_type;
97               switch (this->getDirectionType())
98               {
99               case DirectionType::Forward:
100                   direction_type = "Forward";
101                   break;
102               case DirectionType::Backward:
103                   direction_type = "Backward";
104                   break;
105               }
106               if (name != "")
107                   return fmt::format(R"(<Action:Drive name='{}' speed='{}' direction_type='{}'/>)", name,
      this->getSpeed(), direction_type);
108               else
109                   return fmt::format(R"(<Action:Drive speed='{}' direction_type='{}'/>)",
      this->getSpeed(), direction_type);
110           }
111
112           const int getSpeed() const
113           {
114               return this->speed;
115           }
116
117           const DirectionType getDirectionType() const
118           {
119               return this->direction_type;
120           }
121
```

```
122    private:
123        const int speed;
124        const DirectionType direction_type;
125    };
126 }
127
128 #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.
```
1 #ifndef BEHAVIOUR_TREE_CLOCKDIRECTIONTYPE_HPP
2 #define BEHAVIOUR_TREE_CLOCKDIRECTIONTYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::custom::action
7 {
8     enum class ClockDirectionType
9     {
10        Clockwise,
11        AntiClockwise,
12    };
13 }
14
15 #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.
```
1 #ifndef BEHAVIOUR_TREE_DIRECTIONTYPE_HPP
2 #define BEHAVIOUR_TREE_DIRECTIONTYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::custom::action
7 {
8     enum class DirectionType
9     {
10         Forward,
11         Backward,
12     };
13 }
14
15 #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.
```
1 #ifndef BEHAVIOUR_TREE_SERVOTYPE_HPP
2 #define BEHAVIOUR_TREE_SERVOTYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::custom::action
7 {
8     enum class ServoType
9     {
10         FrontWheels,
11         CameraServo1,
12         CameraServo2,
13     };
14 }
15
16 #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.
```
1 #ifndef BEHAVIOUR_TREE_WHEELTYPE_HPP
2 #define BEHAVIOUR_TREE_WHEELTYPE_HPP
3
4 #pragma once
5
6 namespace behaviour_tree::node::custom::action
7 {
8     enum class WheelType
9     {
10         Left,
11         Right,
12         Both,
13     };
14 }
15
16 #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.](#)

```cpp
1 #ifndef BEHAVIOUR_TREE_PAUSE_EXECUTION_HPP
2 #define BEHAVIOUR_TREE_PAUSE_EXECUTION_HPP
3
4 #pragma once
5
6 #include <chrono>
7 #include <fmt/format.h>
8
9 #include "behaviour_tree/node/custom/CustomNode.hpp"
10
11 #include "behaviour_tree/Context.h"
12
13 namespace behaviour_tree::node::custom::action
14 {
15     class PauseExecution final : public CustomNode
16     {
17     public:
18         PauseExecution(const std::string &name, const int ms) : CustomNode(name), ms(ms)
19         {
20         }
21
22         const static tl::expected<std::shared_ptr<PauseExecution>, std::string> parse(const
    pugi::xml_node &node, const int index, const std::string &name_attribute)
23         {
24             int ms = node.attribute("ms").as_int();
25             if (ms < 0)
26             {
27                 return tl::unexpected(fmt::format(R"(Invalid ms: '{}' |
    Action:PauseExecution:['{}',{}])", ms, name_attribute, index));
28             }
29             return std::make_shared<custom::action::PauseExecution>(
30                 custom::action::PauseExecution(
31                     name_attribute,
32                     ms));
33         }
34
35         void start(std::shared_ptr<Context> context) final override
36         {
37 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
38             this->start_time = std::chrono::steady_clock::now();
39 #endif
40         }
41
42         void finish(std::shared_ptr<Context> context) final override
43         {
44         }
45
46         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
47         {
48 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
49             if (std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::steady_clock::now() -
    this->start_time).count() < this->ms)
50             {
51                 return Status::Running;
52             }
53 #endif
54             return Status::Success;
55         }
56
57         const std::string toString() const final override
58         {
59             const std::string &name = this->getName();
60             if (name != "")
61                 return fmt::format(R"(<Action:PauseExecution name='{}' ms='{}'/>)", name, this->getMS());
62             else
63                 return fmt::format(R"(<Action:PauseExecution ms='{}'/>)", this->getMS());
64         }
65
66         const int getMS() const
67         {
68             return this->ms;
69         }
70
71     private:
72         std::chrono::steady_clock::time_point start_time;
73         const int ms;
74     };
75 }
76
77 #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.
```
1  #ifndef BEHAVIOUR_TREE_PRINT_HPP
2  #define BEHAVIOUR_TREE_PRINT_HPP
3
4  #pragma once
5
6  #include <iostream>
7
8  #include "behaviour_tree/node/custom/CustomNode.hpp"
9
10 #include "behaviour_tree/Context.h"
11 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
12 #include "behaviour_tree/CarContext.hpp"
13 #include <spdlog/spdlog.h>
14 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
15
16 namespace behaviour_tree::node::custom::action
17 {
18     class Print final : public CustomNode
19     {
20     public:
21         Print(const std::string &name, const std::string &text) : CustomNode(name), text(text)
22         {
23         }
24
25         const static tl::expected<std::shared_ptr<Print>, std::string> parse(const pugi::xml_node &node,
    const int index, const std::string &name_attribute)
26         {
27             return std::make_shared<Print>(
28                 Print(
29                     name_attribute,
30                     node.attribute("text").as_string()));
31         }
32
33         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
34         {
35 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
36             std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
37             spdlog::info(this->getText());
38 #else
39             std::cout << this->getText();
40 #endif
```

```
41              return Status::Success;
42          }
43
44          const std::string &getText() const
45          {
46              return this->text;
47          }
48
49          const std::string toString() const final override
50          {
51              const std::string &name = this->getName();
52              if (name != "")
53                  return fmt::format(R"(<Action:Print name='{}' text='{}'/>)", name, this->getText());
54              else
55                  return fmt::format(R"(<Action:Print text='{}'/>)", this->getText());
56          }
57
58      private:
59          const std::string text;
60      };
61 }
62
63 #endif
```

8.74 behaviour_tree_car/include/behaviour_tree/node/custom/action/$\hookleftarrow$ 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.
```
1 #ifndef BEHAVIOUR_TREE_SETANGLE_HPP
2 #define BEHAVIOUR_TREE_SETANGLE_HPP
3
4 #pragma once
5
6 #include "behaviour_tree/node/custom/CustomNode.hpp"
7
8 #include "behaviour_tree/Context.h"
9 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
10 #include "behaviour_tree/CarContext.hpp"
11 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
12
13 #include "enum/ServoType.hpp"
14
15 namespace behaviour_tree::node::custom::action
```

```
16 {
17     class SetAngle final : public CustomNode
18     {
19     public:
20         SetAngle(const std::string &name, const ServoType servo_type, const int angle) :
       CustomNode(name), servo_type(servo_type), angle(angle)
21         {
22         }
23
24         const static tl::expected<std::shared_ptr<SetAngle>, std::string> parse(const pugi::xml_node
       &node, const int index, const std::string &name_attribute)
25         {
26             const tl::expected<ServoType, std::string> maybe_servo_type = [&]()
27             {
28                 const std::string servo_type_attribute = node.attribute("servo_type").as_string();
29                 tl::expected<ServoType, std::string> result;
30                 switch (utils::hash(servo_type_attribute))
31                 {
32                 case utils::hash("FrontWheels"):
33                 {
34                     result = ServoType::FrontWheels;
35                     break;
36                 }
37                 case utils::hash("CameraServo1"):
38                 {
39                     result = ServoType::CameraServo1;
40                     break;
41                 }
42                 case utils::hash("CameraServo2"):
43                 {
44                     result = ServoType::CameraServo2;
45                     break;
46                 }
47                 default:
48                 {
49                     result = tl::unexpected(fmt::format(R"(Invalid servo_type: '{}' |
       Action:SetAngle:['{}',{}])", servo_type_attribute, name_attribute, index));
50                 }
51                 };
52                 return result;
53             }();
54             const int angle = node.attribute("angle").as_int();
55             if (angle < 0 || angle > 180)
56             {
57                 return tl::unexpected(fmt::format(R"(Invalid angle: '{}' | Action:SetAngle:['{}',{}])",
       angle, name_attribute, index));
58             }
59             return std::make_shared<SetAngle>(SetAngle(name_attribute, maybe_servo_type.value(), angle));
60         }
61
62         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
63         {
64 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
65             std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
66             auto car_system = car_context->getCarSystem();
67             const MovementSystem *movement_system = car_system->getMovementSystem();
68             switch (this->getServoType())
69             {
70             case ServoType::FrontWheels:
71                 movement_system->setFrontWheelsAngle(this->getAngle());
72                 break;
73             case ServoType::CameraServo1:
74                 movement_system->setCameraServo1Angle(this->getAngle());
75                 break;
76             case ServoType::CameraServo2:
77                 movement_system->setCameraServo2Angle(this->getAngle());
78                 break;
79             }
80 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
81             return Status::Success;
82         }
83
84         const std::string toString() const final override
85         {
86             std::string servo_type;
87             switch (this->getServoType())
88             {
89             case ServoType::FrontWheels:
90                 servo_type = "FrontWheels";
91                 break;
92             case ServoType::CameraServo1:
93                 servo_type = "CameraServo1";
94                 break;
95             case ServoType::CameraServo2:
96                 servo_type = "CameraServo2";
97                 break;
98             }
```

```
99                const std::string &name = this->getName();
100                if (name != "")
101                    return fmt::format(R"(<Action:SetAngle name='{}' servo_type='{}' angle='{}'/>)", name,
       servo_type, this->getAngle());
102                else
103                    return fmt::format(R"(<Action:SetAngle servo_type='{}' angle='{}'/>)", servo_type,
       this->getAngle());
104            }
105
106        const ServoType getServoType() const
107        {
108            return this->servo_type;
109        }
110
111        const int getAngle() const
112        {
113            return this->angle;
114        }
115
116    private:
117        const int angle;
118        const ServoType servo_type;
119    };
120 }
121
122 #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.
```
1 #ifndef BEHAVIOUR_TREE_SETSPEED_HPP
2 #define BEHAVIOUR_TREE_SETSPEED_HPP
3
4 #pragma once
5
6 #include <fmt/format.h>
7
8 #include "behaviour_tree/node/custom/CustomNode.hpp"
9
10 #include "behaviour_tree/Context.h"
11 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
```

```
12 #include "behaviour_tree/CarContext.hpp"
13 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
14
15 #include "enum/WheelType.hpp"
16
17 namespace behaviour_tree::node::custom::action
18 {
19     class SetSpeed final : public CustomNode
20     {
21     public:
22         SetSpeed(const std::string &name, const WheelType wheel_type, const int speed) :
    CustomNode(name), wheel_type(wheel_type), speed(speed)
23         {
24         }
25
26         const static tl::expected<std::shared_ptr<SetSpeed>, std::string> parse(const pugi::xml_node
    &node, const int index, const std::string &name_attribute)
27         {
28             const tl::expected<WheelType, std::string> maybe_wheel_type = [&]()
29             {
30                 const std::string wheel_type_attribute = node.attribute("wheel_type").as_string();
31                 tl::expected<WheelType, std::string> result;
32                 switch (utils::hash(wheel_type_attribute))
33                 {
34                 case utils::hash("Left"):
35                 {
36                     result = WheelType::Left;
37                     break;
38                 }
39                 case utils::hash("Right"):
40                 {
41                     result = WheelType::Right;
42                     break;
43                 }
44                 case utils::hash("Both"):
45                 {
46                     result = WheelType::Both;
47                     break;
48                 }
49                 default:
50                 {
51                     result = tl::unexpected(fmt::format(R"(Invalid wheel_type: '{}' |
    Action:SetSpeed:['{}',{}])", wheel_type_attribute, name_attribute, index));
52                     break;
53                 }
54                 };
55                 return result;
56             }();
57             if (!maybe_wheel_type.has_value())
58             {
59                 return tl::make_unexpected(maybe_wheel_type.error());
60             }
61             const int speed = node.attribute("speed").as_int();
62             if (speed < 0 || speed > 100)
63             {
64                 return tl::unexpected(fmt::format(R"(Invalid speed: '{}' | Action:SetSpeed:['{}',{}])",
    speed, name_attribute, index));
65             }
66             return std::make_shared<SetSpeed>(SetSpeed(name_attribute, maybe_wheel_type.value(), speed));
67         }
68
69         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
70         {
71 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
72             std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
73             auto car_system = car_context->getCarSystem();
74             const MovementSystem *movement_system = car_system->getMovementSystem();
75             switch (this->getWheelType())
76             {
77             case WheelType::Left:
78                 movement_system->setRearLeftWheelSpeed(this->getSpeed());
79                 break;
80             case WheelType::Right:
81                 movement_system->setRearRightWheelSpeed(this->getSpeed());
82                 break;
83             case WheelType::Both:
84                 movement_system->setRearWheelsSpeed(this->getSpeed());
85                 break;
86             }
87 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
88             return Status::Success;
89         }
90
91         const std::string toString() const final override
92         {
93             std::string wheel_type;
94             switch (this->getWheelType())
```

```
 95                {
 96            case WheelType::Left:
 97                wheel_type = "Forward";
 98                break;
 99            case WheelType::Right:
100                wheel_type = "Backward";
101                break;
102             case WheelType::Both:
103                wheel_type = "Both";
104                break;
105            }
106            const std::string &name = this->getName();
107            if (name != "")
108                return fmt::format(R"(<Action:SetSpeed name='{}' wheel_type='{}' speed='{}'/>)", name,
     wheel_type, this->getSpeed());
109            else
110                return fmt::format(R"(<Action:SetSpeed wheel_type='{}' speed='{}'/>)", wheel_type,
     this->getSpeed());
111        }
112
113        const WheelType getWheelType() const
114        {
115            return this->wheel_type;
116        }
117
118        const int getSpeed() const
119        {
120            return this->speed;
121        }
122
123    private:
124        const int speed;
125        const WheelType wheel_type;
126    };
127 }
128
129 #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.
```
1 #ifndef BEHAVIOUR_TREE_SETWHEELDIRECTION_HPP
2 #define BEHAVIOUR_TREE_SETWHEELDIRECTION_HPP
3
4 #pragma once
5
6 #include "behaviour_tree/node/custom/CustomNode.hpp"
7
8 #include "behaviour_tree/Context.h"
9 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
10 #include "behaviour_tree/CarContext.hpp"
11 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
12
13 #include "enum/DirectionType.hpp"
14 #include "enum/WheelType.hpp"
15
16 namespace behaviour_tree::node::custom::action
17 {
18     class SetWheelDirection final : public CustomNode
19     {
20     public:
21         SetWheelDirection(const std::string &name, const WheelType wheel_type, const DirectionType
    direction_type) : CustomNode(name), wheel_type(wheel_type), direction_type(direction_type)
22         {
23         }
24
25         const static tl::expected<std::shared_ptr<SetWheelDirection>, std::string> parse(const
    pugi::xml_node &node, const int index, const std::string &name_attribute)
26         {
27             const tl::expected<WheelType, std::string> maybe_wheel_type = [&]()
28             {
29                 const std::string wheel_type_attribute = node.attribute("wheel_type").as_string();
30                 tl::expected<WheelType, std::string> result;
31                 switch (utils::hash(wheel_type_attribute))
32                 {
33                 case utils::hash("Left"):
34                 {
35                     result = WheelType::Left;
36                     break;
37                 }
38                 case utils::hash("Right"):
39                 {
40                     result = WheelType::Right;
41                     break;
42                 }
43                 case utils::hash("Both"):
44                 {
45                     result = WheelType::Both;
46                     break;
47                 }
48                 default:
49                 {
50                     result = tl::unexpected(fmt::format(R"(Invalid wheel_type: '{}' |
    Action:SetWheelDirection:['{}',{}])", wheel_type_attribute, name_attribute, index));
51                     break;
52                 }
53                 };
54                 return result;
55             }();
56             if (!maybe_wheel_type.has_value())
57             {
58                 return tl::make_unexpected(maybe_wheel_type.error());
59             }
60             const tl::expected<DirectionType, std::string> maybe_direction_type =
61                 [&]()
62             {
63                 const std::string direction_type_attribute =
    node.attribute("direction_type").as_string();
64                 tl::expected<DirectionType, std::string> result;
65                 switch (utils::hash(direction_type_attribute))
66                 {
67                 case utils::hash("Forward"):
68                 {
69                     result = DirectionType::Forward;
70                 }
71                 case utils::hash("Backward"):
72                 {
73                     result = DirectionType::Backward;
74                 }
75                 default:
76                 {
77                     result = tl::make_unexpected(fmt::format(R"(Invalid direction_type: '{}' |
    Action:Drive:['{}',{}])", direction_type_attribute, name_attribute, index));
```

```
78                     }
79                 };
80                 return result;
81             }();
82             if (!maybe_direction_type.has_value())
83             {
84                 return tl::make_unexpected(maybe_direction_type.error());
85             }
86             return std::make_shared<SetWheelDirection>(SetWheelDirection(name_attribute,
     maybe_wheel_type.value(), maybe_direction_type.value()));
87         }
88
89         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
90         {
91 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
92             std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
93             auto car_system = car_context->getCarSystem();
94             const MovementSystem *movement_system = car_system->getMovementSystem();
95             switch (this->getDirectionType())
96             {
97             case DirectionType::Forward:
98             {
99                 switch (this->getWheelType())
100                {
101                case WheelType::Left:
102                {
103                    movement_system->setRearLeftWheelDirectionToForward();
104                    break;
105                }
106                case WheelType::Right:
107                {
108                    movement_system->setRearRightWheelDirectionToForward();
109                    break;
110                }
111                case WheelType::Both:
112                {
113                    movement_system->setRearWheelsDirectionToForward();
114                    break;
115                }
116                }
117                break;
118            }
119            case DirectionType::Backward:
120            {
121                switch (this->getWheelType())
122                {
123                case WheelType::Left:
124                {
125                    movement_system->setRearLeftWheelDirectionToBackward();
126                    break;
127                }
128                case WheelType::Right:
129                {
130                    movement_system->setRearRightWheelDirectionToBackward();
131                    break;
132                }
133                case WheelType::Both:
134                {
135                    movement_system->setRearWheelsDirectionToBackward();
136                    break;
137                }
138                }
139                break;
140            }
141            }
142 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
143            return Status::Success;
144        }
145
146        const std::string toString() const final override
147        {
148            std::string direction_type;
149            switch (this->getDirectionType())
150            {
151            case DirectionType::Forward:
152                direction_type = "Forward";
153                break;
154            case DirectionType::Backward:
155                direction_type = "Backward";
156                break;
157            }
158            std::string wheel_type;
159            switch (this->getWheelType())
160            {
161            case WheelType::Left:
162                wheel_type = "Forward";
163                break;
```

```
164              case WheelType::Right:
165                  wheel_type = "Backward";
166                  break;
167              case WheelType::Both:
168                  wheel_type = "Both";
169                  break;
170              }
171              const std::string &name = this->getName();
172              if (name != "")
173                  return fmt::format(R"(<Action:SetWheelDirection name='{}' direction_type='{}'
     wheel_type='{}'/>)", name, direction_type, wheel_type);
174              else
175                  return fmt::format(R"(<Action:SetWheelDirection direction_type='{}' wheel_type='{}'/>)",
     direction_type, wheel_type);
176          }
177
178          const DirectionType getDirectionType() const
179          {
180              return this->direction_type;
181          }
182
183          const WheelType getWheelType() const
184          {
185              return this->wheel_type;
186          }
187
188      private:
189          const DirectionType direction_type;
190          const WheelType wheel_type;
191      };
192 }
193
194 #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.
```
1 #ifndef BEHAVIOUR_TREE_TURN_HPP
2 #define BEHAVIOUR_TREE_TURN_HPP
3
4 #pragma once
5
6 #include "behaviour_tree/node/custom/CustomNode.hpp"
7
```

```
8 #include "behaviour_tree/Context.h"
9 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
10 #include "behaviour_tree/CarContext.hpp"
11 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
12
13
14 namespace behaviour_tree::node::custom::action
15 {
16     class Turn final : public CustomNode
17     {
18     public:
19         Turn(const std::string &name, const int angle) : CustomNode(name), angle(angle)
20         {
21         }
22
23         const static tl::expected<std::shared_ptr<Turn>, std::string> parse(const pugi::xml_node &node,
     const int index, const std::string &name_attribute)
24         {
25             const int angle = node.attribute("angle").as_int();
26             if (angle < 0 || angle > 180)
27             {
28                 return tl::unexpected(fmt::format(R"(Invalid angle: '{}' | Action:SetAngle:['{}',{}])",
     angle, name_attribute, index));
29             }
30             return std::make_shared<Turn>(Turn(name_attribute, angle));
31         }
32
33         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
34         {
35 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
36             std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
37             auto car_system = car_context->getCarSystem();
38             const MovementSystem *movement_system = car_system->getMovementSystem();
39             movement_system->setFrontWheelsAngle(this->getAngle());
40 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
41             return Status::Success;
42         }
43
44         const std::string toString() const final override
45         {
46             const std::string &name = this->getName();
47             if (name != "")
48                 return fmt::format(R"(<Action:Turn name='{}' angle='{}'/>)", name, this->getAngle());
49             else
50                 return fmt::format(R"(<Action:Turn angle='{}'/>)", this->getAngle());
51         }
52
53         const int getAngle() const
54         {
55             return this->angle;
56         }
57
58     private:
59         const int angle;
60     };
61 }
62
63 #endif
```

8.82 behaviour_tree_car/include/behaviour_tree/node/custom/Car↩ CustomNodeParser.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.
```
1 #ifndef BEHAVIOUR_TREE_CAR_TASK_NODE_PARSER_HPP
2 #define BEHAVIOUR_TREE_CAR_TASK_NODE_PARSER_HPP
3
4 #pragma once
5
6 #include <string>
7 #include <memory>
8
9 #include <tl/expected.hpp>
10 #include <pugixml.hpp>
11 #include <fmt/format.h>
12
13 #include "behaviour_tree/BehaviourTree.hpp"
14 #include "behaviour_tree/node/custom/CustomNode.hpp"
15
16 #include "behaviour_tree/node/custom/CustomNodeParser.hpp"
17
18 #include "action/PauseExecution.hpp"
19
20 #include "action/Print.hpp"
21
22 #include "action/Drive.hpp"
23 #include "action/Turn.hpp"
24 #include "action/SetSpeed.hpp"
25 #include "action/SetWheelDirection.hpp"
26 #include "action/SetAngle.hpp"
27
28 #include "condition/SucceedOnAverageNearbyScan.hpp"
29 #include "condition/SucceedOnAnyNearbyScan.hpp"
30 #include "condition/SucceedOnAverageColour.hpp"
31
32 #include "utils/Utility.hpp"
33
34 namespace behaviour_tree::node::custom
35 {
36     class CarCustomNodeParser : public custom::CustomNodeParser
37     {
38     public:
39         CarCustomNodeParser() {}
40
41         tl::expected<std::shared_ptr<custom::CustomNode>, std::string> parseCustomNode(const
    pugi::xml_node &node, const int index) override
42         {
43             const std::string name_attribute = node.attribute("node_name").as_string();
44             const std::string node_name = node.name();
45             switch (utils::hash(node_name))
```

```
46              {
47              case utils::hash("Action:PauseExecution"):
48                  return custom::action::PauseExecution::parse(node, index, name_attribute);
49              case utils::hash("Action:Print"):
50                  return custom::action::Print::parse(node, index, name_attribute);
51              case utils::hash("Action:Drive"):
52                  return custom::action::Drive::parse(node, index, name_attribute);
53              case utils::hash("Action:Turn"):
54                  return custom::action::Turn::parse(node, index, name_attribute);
55              case utils::hash("Action:SetSpeed"):
56                  return custom::action::SetSpeed::parse(node, index, name_attribute);
57              case utils::hash("Action:SetAngle"):
58                  return custom::action::SetAngle::parse(node, index, name_attribute);
59              case utils::hash("Action:SetWheelDirection"):
60                  return custom::action::SetWheelDirection::parse(node, index, name_attribute);
61              case utils::hash("Condition:SucceedOnAverageNearbyScan"):
62                  return custom::condition::SucceedOnAverageNearbyScan::parse(node, index, name_attribute);
63              case utils::hash("Condition:SucceedOnAnyNearbyScan"):
64                  return custom::condition::SucceedOnAnyNearbyScan::parse(node, index, name_attribute);
65              case utils::hash("Condition:SucceedOnAverageColour"):
66                  return custom::condition::SucceedOnAverageColour::parse(node, index, name_attribute);
67              default:
68              {
69                  return tl::unexpected(fmt::format(R"(Invalid custom node type: '{}' | {}:['{}',{}])",
70      node_name, node_name, name_attribute, index));
71              }
72          }
73      };
74  }
75
76  #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.

```cpp
1 #ifndef BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_ANY_SCAN_HPP
2 #define BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_ANY_SCAN_HPP
3
4 #pragma once
5
6 #include "behaviour_tree/node/custom/CustomNode.hpp"
7
8 #include "behaviour_tree/Context.h"
9 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
10 #include "behaviour_tree/CarContext.hpp"
11 #include <spdlog/spdlog.h>
12 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
13
14 #include "behaviour_tree/global/distance.h"
15
16 namespace behaviour_tree::node::custom::condition
17 {
18     class SucceedOnAnyNearbyScan final : public CustomNode
19     {
20     public:
21         SucceedOnAnyNearbyScan(const std::string &name, const int min_angle, const int max_angle, const
     double cm) : CustomNode(name),
22                 min_angle(min_angle),
23                 max_angle(max_angle),
24                 cm(cm),
25                 minimum_distance_unit(cm * global::CM_TO_DISTANCE)
26         {
27         }
28
29         const static tl::expected<std::shared_ptr<SucceedOnAnyNearbyScan>, std::string> parse(const
     pugi::xml_node &node, const int index, const std::string &name_attribute)
30         {
31             const int min_angle = node.attribute("min_angle").as_int();
32             if (min_angle < 0 || min_angle > 360)
33             {
34                 return tl::unexpected(fmt::format(R"(Invalid min_angle: '{}' |
     Condition:SucceedOnAnyNearbyScan:['{}',{}])", min_angle, name_attribute, index));
35             }
36             const int max_angle = node.attribute("max_angle").as_int();
37             if (max_angle < 0 || max_angle > 360)
38             {
39                 return tl::unexpected(fmt::format(R"(Invalid max_angle: '{}' |
     Condition:SucceedOnAnyNearbyScan:['{}',{}])", max_angle, name_attribute, index));
40             }
41             const double cm = node.attribute("cm").as_double();
42             if (cm < 0)
43             {
44                 return tl::unexpected(fmt::format(R"(Invalid cm: '{}' |
     Condition:SucceedOnAnyNearbyScan:['{}',{}])", cm, name_attribute, index));
45             }
46             return std::make_shared<SucceedOnAnyNearbyScan>(
47                 SucceedOnAnyNearbyScan(
48                     name_attribute,
49                     min_angle,
50                     max_angle,
51                     cm));
52         }
53
54         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
55         {
56 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
57             std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
58             auto car_system = car_context->getCarSystem();
59             std::vector<Measure> scan_data =
     car_system->getDeviceManager()->getLidarDevice()->getScanData();
60             for (auto &measure : scan_data)
61             {
62                 if (measure.angle > this->getMinAngle() && measure.angle < this->getMaxAngle())
63                 {
64                     if (measure.distance < this->getMinimumDistanceUnit())
65                     {
66                         return Status::Success;
67                     }
68                 }
69             }
70 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
71             return Status::Failure;
72         }
```

```
73
74          const int getMinAngle() const
75          {
76              return this->min_angle;
77          }
78
79          const int getMaxAngle() const
80          {
81              return this->max_angle;
82          }
83
84          const double getMinimumDistanceUnit() const
85          {
86              return this->minimum_distance_unit;
87          }
88
89          const double getCentimeters() const
90          {
91              return this->cm;
92          }
93
94          const std::string toString() const final override
95          {
96              const std::string &name = this->getName();
97              if (name != "")
98                  return fmt::format(R"(<Condition:SucceedOnAnyNearbyScan name='{}' min_angle='{}'
     max_angle='{}' cm='{}'/>)", name, this->getMinAngle(), this->getMaxAngle(), this->getCentimeters());
99              else
100                 return fmt::format(R"(<Condition:SucceedOnAnyNearbyScan min_angle='{}' max_angle='{}'
     cm='{}'/>)", this->getMinAngle(), this->getMaxAngle(), this->getCentimeters());
101         }
102
103   private:
104         const int min_angle;
105         const int max_angle;
106
107         const double cm;
108
109         const double minimum_distance_unit;
110     };
111 }
112
113 #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.
```cpp
1  #ifndef BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_COLOUR_HPP
2  #define BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_COLOUR_HPP
3
4  #pragma once
5
6  #include "behaviour_tree/node/custom/CustomNode.hpp"
7
8  #include "behaviour_tree/Context.h"
9  #ifndef BEHAVIOUR_TREE_DISABLE_RUN
10 #include "behaviour_tree/CarContext.hpp"
11 #include <spdlog/spdlog.h>
12 #include <opencv2/opencv.hpp>
13 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
14
15 namespace behaviour_tree::node::custom::condition
16 {
17     class SucceedOnAverageColour final : public CustomNode
18     {
19     public:
20         SucceedOnAverageColour(const std::string &name, const std::string hex_colour, const double
       tolerance) : CustomNode(name),
21
       hex_colour(hex_colour),
22
       tolerance(tolerance)
23         {
24         }
25
26         const static tl::expected<std::shared_ptr<SucceedOnAverageColour>, std::string> parse(const
       pugi::xml_node &node, const int index, const std::string &name_attribute)
27         {
28             std::string hex_colour = node.attribute("hex_colour").as_string();
29             if (hex_colour.compare(0, 1, "#"))
30             {
31                 hex_colour = "#" + hex_colour;
32             }
33             if (hex_colour.size() != 7)
34             {
35                 return tl::unexpected(fmt::format(R"(Invalid hex_colour: '{}' |
       Condition:SucceedOnAverageColour:['{}',{}])", hex_colour, name_attribute, index));
36             }
37             for (int i = 1; i < hex_colour.size(); i++)
38             {
39                 if (!std::isxdigit(hex_colour[i]))
40                 {
41                     return tl::unexpected(fmt::format(R"(Invalid hex_colour: '{}' |
       Condition:SucceedOnAverageColour:['{}',{}])", hex_colour, name_attribute, index));
42                 }
43             }
44             const double tolerance = node.attribute("tolerance").as_double();
45             if (tolerance < 0 || tolerance > 100)
46             {
47                 return tl::unexpected(fmt::format(R"(Invalid tolerance: '{}' |
       Condition:SucceedOnAverageColour:['{}',{}])", tolerance, name_attribute, index));
48             }
49             return std::make_shared<SucceedOnAverageColour>(
50                 SucceedOnAverageColour(
51                     name_attribute,
52                     hex_colour,
53                     tolerance));
54         }
55
56         const Status run(const int tick_count, std::shared_ptr<Context> context) final override
57         {
58             // Following made by ChatGPT
59 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
60             std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
61             auto car_system = car_context->getCarSystem();
62
63             std::string frame_buffer =
       car_system->getDeviceManager()->getCameraDevice()->getFrameBuffer();
64             std::vector<uchar> buffer(frame_buffer.begin(), frame_buffer.end());
65             cv::Mat frame = cv::imdecode(buffer, cv::IMREAD_COLOR);
66             if (frame.empty())
67             {
68                 spdlog::error("Failed to decode frame buffer");
69                 return Status::Failure;
70             }
71
72             cv::Scalar avg_color = cv::mean(frame);
73             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]));
```

```
74
75                int color_diff = calculateColorDifference(avg_color_hex, this->hex_colour);
76                int max_color_diff = calculateMaxColorDifference(this->tolerance);
77
78                if (color_diff <= max_color_diff)
79                {
80                    return Status::Success;
81                }
82                else
83                {
84                    return Status::Failure;
85                }
86 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
87            return Status::Failure;
88        }
89
90        // Following made by ChatGPT
91 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
92        // Function to calculate the color difference between two hex colors
93        int calculateColorDifference(const std::string &color1, const std::string &color2)
94        {
95            // Convert hex strings to RGB values
96            cv::Scalar rgb_color1 = hexToRGB(color1);
97            cv::Scalar rgb_color2 = hexToRGB(color2);
98
99            // Calculate the Euclidean distance between the two colors in RGB space
100            return static_cast<int>(cv::norm(rgb_color1, rgb_color2));
101        }
102
103        // Function to convert a hex color string to RGB values
104        cv::Scalar hexToRGB(const std::string &hexColor)
105        {
106            std::string hex = hexColor.substr(1); // remove the '#' character
107            int r, g, b;
108            std::istringstream(hex.substr(0, 2)) >> std::hex >> r;
109            std::istringstream(hex.substr(2, 2)) >> std::hex >> g;
110            std::istringstream(hex.substr(4, 2)) >> std::hex >> b;
111            return cv::Scalar(b, g, r);
112        }
113
114        // Function to calculate the maximum allowable color difference based on the percentage
     threshold
115        int calculateMaxColorDifference(double percentage)
116        {
117            // Assuming percentage represents the acceptable deviation from the specified color
118            // We can calculate the maximum allowable color difference based on this percentage
119            // For example, if percentage is 10, it means 10% deviation is acceptable
120
121            // Max color difference will be based on the maximum possible Euclidean distance between two
     colors in RGB space
122            // For simplicity, let's assume the maximum distance is between pure black and pure white
123            cv::Scalar black(0, 0, 0);
124            cv::Scalar white(255, 255, 255);
125            int max_color_diff = static_cast<int>(cv::norm(black, white));
126
127            // Now, we scale the maximum color difference based on the percentage
128            return static_cast<int>(max_color_diff * (percentage / 100.0));
129        }
130 #endif // !BEHAVIOUR_TREE_DISABLE_RUNss
131
132        const std::string getHexColour() const
133        {
134            return this->hex_colour;
135        }
136
137        const double getTolerance() const
138        {
139            return this->tolerance;
140        }
141
142        const std::string toString() const final override
143        {
144            const std::string &name = this->getName();
145            if (name != "")
146                return fmt::format(R"(<Condition:SucceedOnAverageColour name='{}' hex_colour='{}'
     tolerance='{}'/>)", name, this->getHexColour(), this->getTolerance());
147            else
148                return fmt::format(R"(<Condition:SucceedOnAverageColour hex_colour='{}'
     tolerance='{}'/>)", this->getHexColour(), this->getTolerance());
149        }
150
151    private:
152        const std::string hex_colour;
153        const double tolerance;
154    };
155 }
156
```

```
157 #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.
```
1 #ifndef BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_NEARBY_SCAN_HPP
2 #define BEHAVIOUR_TREE_SUCCEED_ON_AVERAGE_NEARBY_SCAN_HPP
3
4 #pragma once
5
6 #include "behaviour_tree/node/custom/CustomNode.hpp"
7
8 #include "behaviour_tree/Context.h"
9 #ifndef BEHAVIOUR_TREE_DISABLE_RUN
10 #include "behaviour_tree/CarContext.hpp"
11 #include <spdlog/spdlog.h>
12 #endif // !BEHAVIOUR_TREE_DISABLE_RUN
13
14 #include "behaviour_tree/global/distance.h"
15
16 namespace behaviour_tree::node::custom::condition
17 {
18     class SucceedOnAverageNearbyScan final : public CustomNode
19     {
20     public:
21         SucceedOnAverageNearbyScan(const std::string &name, const int min_angle, const int max_angle,
    const double cm, const int smallest_measure_amount_used) : CustomNode(name),
22                                                         min_angle(min_angle),
23                                                         max_angle(max_angle),
24                                                         cm(cm),
25
    smallest_measure_amount_used(smallest_measure_amount_used),
```

```
26
                                                                    average_distance_unit(cm *
     global::CM_TO_DISTANCE)
27          {
28          }
29
30          const static tl::expected<std::shared_ptr<SucceedOnAverageNearbyScan>, std::string> parse(const
     pugi::xml_node &node, const int index, const std::string &name_attribute)
31          {
32              const int min_angle = node.attribute("min_angle").as_int();
33              if (min_angle < 0 || min_angle > 360)
34              {
35                  return tl::unexpected(fmt::format(R"(Invalid min_angle: '{}' |
     Condition:SucceedOnAverageNearbyScan:['{}',{}])", min_angle, name_attribute, index));
36              }
37              const int max_angle = node.attribute("max_angle").as_int();
38              if (max_angle < 0 || max_angle > 360)
39              {
40                  return tl::unexpected(fmt::format(R"(Invalid max_angle: '{}' |
     Condition:SucceedOnAverageNearbyScan:['{}',{}])", max_angle, name_attribute, index));
41              }
42              const double cm = node.attribute("cm").as_double();
43              if (cm < 0)
44              {
45                  return tl::unexpected(fmt::format(R"(Invalid cm: '{}' |
     Condition:SucceedOnAverageNearbyScan:['{}',{}])", cm, name_attribute, index));
46              }
47              const int minimum_measure_amount_used =
     node.attribute("minimum_measure_amount_used").as_int();
48              if (minimum_measure_amount_used < 0)
49              {
50                  return tl::unexpected(fmt::format(R"(Invalid minimum_measure_amount_used: '{}' |
     Condition:SucceedOnAverageNearbyScan:['{}',{}])", minimum_measure_amount_used, name_attribute,
     index));
51              }
52              return std::make_shared<SucceedOnAverageNearbyScan>(
53                  SucceedOnAverageNearbyScan(
54                      name_attribute,
55                      min_angle,
56                      max_angle,
57                      cm,
58                      minimum_measure_amount_used));
59          }
60
61          const Status run(const int tick_count, std::shared_ptr<Context> context) final override
62          {
63  #ifndef BEHAVIOUR_TREE_DISABLE_RUN
64              std::shared_ptr<CarContext> car_context = std::dynamic_pointer_cast<CarContext>(context);
65              auto car_system = car_context->getCarSystem();
66              double total_distance = 0.0;
67              int angles_between_count = 0;
68              std::vector<Measure> scan_data =
     car_system->getDeviceManager()->getLidarDevice()->getScanData();
69              if (this->smallest_measure_amount_used > 0)
70              {
71                  std::vector<Measure> smallest_measures;
72                  std::partial_sort(scan_data.begin(), scan_data.begin() +
     this->smallest_measure_amount_used, scan_data.end(),
73                                    [](const Measure &a, const Measure &b)
74                                    { return a.distance < b.distance; });
75                  smallest_measures.insert(smallest_measures.end(), scan_data.begin(), scan_data.begin() +
     this->smallest_measure_amount_used);
76                  scan_data = smallest_measures;
77              }
78              for (auto &measure : scan_data)
79              {
80                  if (measure.angle > this->getMinAngle() && measure.angle < this->getMaxAngle())
81                  {
82                      total_distance += measure.distance;
83                      ++angles_between_count;
84                  }
85              }
86              if (angles_between_count > 0)
87              {
88                  double average_distance_unit = total_distance / angles_between_count;
89                  if (average_distance_unit < this->getAverageDistanceUnit())
90                  {
91                      return Status::Success;
92                  }
93              }
94  #endif // !BEHAVIOUR_TREE_DISABLE_RUN
95              return Status::Failure;
96          }
97
98          const int getMinAngle() const
99          {
100             return this->min_angle;
```

```
101        }
102
103        const int getMaxAngle() const
104        {
105            return this->max_angle;
106        }
107
108        const double getAverageDistanceUnit() const
109        {
110            return this->average_distance_unit;
111        }
112
113        const double getCentimeters() const
114        {
115            return this->cm;
116        }
117
118        const int getSmallestMeasureAmountUsed() const
119        {
120            return this->smallest_measure_amount_used;
121        }
122
123        const std::string toString() const final override
124        {
125            const std::string &name = this->getName();
126            if (name != "")
127                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());
128            else
129                return fmt::format(R"(<Condition:SucceedOnAverageNearbyScan min_angle='{}'
    max_angle='{}' cm='{}' smallest_measure_amount_used='{}'/>)", this->getMinAngle(),
    this->getMaxAngle(), this->getCentimeters(), this->getSmallestMeasureAmountUsed());
130        }
131
132    private:
133        const int min_angle;
134        const int max_angle;
135
136        const double cm;
137
138        const int smallest_measure_amount_used;
139
140        const double average_distance_unit;
141    };
142 }
143
144 #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.
```
1 #ifndef RPLIDAR_H
2 #define RPLIDAR_H
3
4 #pragma once
5
6 // Made with the help of ChatGPT
7
8
9 /*
10 #include <RPLidar.h>
11 #include <memory>
12 #include <spdlog/spdlog.h>
13
14 int main()
15 {
16     using namespace rplidar;
17     spdlog::set_level(spdlog::level::debug);
```

```
18
19      // auto lidar_result = RPLidar::create("/dev/ttyUSB0"); // For Linux
20      // auto lidar_result = RPLidar::create("COM3"); // For Windows
21      if (!lidar_result.has_value())
22      {
23          std::cout « "Unable to open lidar\n";
24          return 0;
25      }
26      auto &lidar = lidar_result.value();
27      std::cout « "Connected to lidar\n";
28
29      auto info_result = lidar->get_info();
30      if (!info_result.has_value())
31      {
32          std::cout « "Unable to get value of get_info() " « info_result.error();
33          return 0;
34      }
35      auto &info = info_result.value();
36      std::cout « fmt::format("model: {}, firmware: ({}, {}), hardware: {}, serialnumber: {}\n",
        info.model, info.firmware.first, info.firmware.second, info.hardware, info.serialNumber);
37
38      auto health_result = lidar->get_health();
39      if (!health_result.has_value())
40      {
41          std::cout « "Unable to get value of get_health() "« health_result.error();
42          return 0;
43      }
44      auto &health = health_result.value();
45      std::cout « fmt::format("({}, {})\n", health.status, health.errorCode);
46
47      std::function<std::vector<Measure>()> scanGenerator = lidar->iter_scans();
48      for (int i = 0; i < 10; i++)
49      {
50          std::vector<Measure> scan = scanGenerator();
51          std::cout « "Got " « scan.size() « " Measures!\n";
52          for (const Measure &measure : scan)
53          {
54              // Access individual measurements in the scan
55              bool newScan = measure.newScan;
56              int quality = measure.quality;
57              float angle = measure.angle;
58              float distance = measure.distance;
59          }
60      }
61
62      lidar->stop();
63      lidar->stop_motor();
64      lidar->disconnect();
65      return 1;
66 }
67
68 */
69
70 #include <string>
71 #include <stdint.h>
72
73 #include <serial/serial.h>
74
75 #include <iostream>
76 #include <map>
77 #include <vector>
78
79 // std::this_thread::sleep_for(std::chrono::milliseconds(1));
80 #include <chrono>
81 #include <thread>
82
83 // convertToHexString
84 #include <sstream>
85 #include <iomanip>
86
87 // iter_measures
88 #include <functional>
89
90 #include <tuple>
91 #include <stdexcept>
92
93 #include <memory>
94
95 #include <spdlog/spdlog.h>
96 #include <spdlog/fmt/bin_to_hex.h>
97
98 #include <tl/expected.hpp>
99
100 namespace rplidar
101 {
102      constexpr uint8_t SYNC_BYTE = 0xA5;
103      constexpr uint8_t SYNC_BYTE2 = 0x5A;
```

```
104
105     constexpr uint8_t GET_INFO_BYTE = 0x50;
106     constexpr uint8_t GET_HEALTH_BYTE = 0x52;
107
108     constexpr uint8_t STOP_BYTE = 0x25;
109     constexpr uint8_t RESET_BYTE = 0x40;
110
111     enum ScanType
112     {
113         NORMAL = 0,
114         FORCE,
115         EXPRESS
116     };
117
118     struct ScanInfo
119     {
120         int currently_scanning;
121         int dsize;
122         ScanType type;
123     };
124
125     static std::map<ScanType, std::map<std::string, uint8_t» SCAN_TYPE = {
126         {ScanType::NORMAL, {{"byte", 0x20}, {"response", 129}, {"size", 5}}},
127         {ScanType::FORCE, {{"byte", 0x21}, {"response", 129}, {"size", 5}}},
128         {ScanType::EXPRESS, {{"byte", 0x82}, {"response", 130}, {"size", 84}}}};
129
130     constexpr int DESCRIPTOR_LEN = 7;
131     constexpr int INFO_LEN = 20;
132     constexpr int HEALTH_LEN = 3;
133
134     constexpr int INFO_TYPE = 4;
135     constexpr int HEALTH_TYPE = 6;
136
137     constexpr int MAX_MOTOR_PWM = 1023;
138     constexpr int DEFAULT_MOTOR_PWM = 660;
139     constexpr uint8_t SET_PWM_BYTE = 0xF0;
140
141     static std::map<int, std::string> HEALTH_STATUSES = {
142         {0, "Good"},
143         {1, "Warning"},
144         {2, "Error"}};
145
146     struct DeviceInfo
147     {
148         uint8_t model;
149         std::pair<uint8_t, uint8_t> firmware;
150         uint8_t hardware;
151         std::string serialNumber;
152     };
153
158     struct HealthInfo
159     {
164         std::string status;
169         int errorCode;
170     };
171
172     struct Measure
173     {
174         bool newScan;
175         int quality;
176         double angle;
177         double distance;
178     };
179
180     class ExpressPacket
181     {
182     public:
183         static const uint8_t sync1 = 0xa;
184         static const uint8_t sync2 = 0x5;
185
186         ExpressPacket(std::vector<uint8_t> data)
187         {
188             if ((data[0] » 4) != sync1 || (data[1] » 4) != sync2)
189             {
190                 throw std::invalid_argument("try to parse corrupted data");
191             }
192
193             uint8_t checksum = 0;
194             for (size_t i = 2; i < data.size(); i++)
195             {
196                 checksum ^= data[i];
197             }
198
199             if (checksum != ((data[0] & 0x0F) + ((data[1] & 0x0F) « 4)))
200             {
201                 throw std::invalid_argument("Invalid checksum");
202             }
```

```
203
204             new_scan = (data[2] >> 7) & 0x01;
205             start_angle = static_cast<float>((data[1] & 0x0F) << 8 | data[2]) / 64.0f;
206
207             for (size_t i = 4; i < data.size(); i += 5)
208             {
209                 distance.push_back(((data[i + 1] >> 2) & 0x3F) | ((data[i] & 0x3F) << 6));
210                 angle.push_back(((((data[i + 3] & 0x0F) + ((data[i + 1] & 0x01) << 4)) / 8.0f) *
      getSign(data[i + 1]));
211                 distance.push_back(((data[i + 2] >> 2) & 0x3F) | ((data[i + 1] & 0x3F) << 6));
212                 angle.push_back(((((data[i + 3] >> 4) & 0x0F) + ((data[i + 2] & 0x01) << 4)) / 8.0f *
      getSign(data[i + 2]));
213             }
214         }
215
216         static int getSign(uint8_t value)
217         {
218             return (value & 0x02) ? -1 : 1;
219         }
220
221     public:
222         std::vector<uint16_t> distance;
223         std::vector<float> angle;
224         bool new_scan;
225         float start_angle;
226     };
231     class RPLidar
232     {
233     public:
234         static tl::expected<std::unique_ptr<RPLidar>, std::string> create(const std::string &port,
      uint32_t baudrate = 115200U) noexcept
235         {
236             try
237             {
238                 // Timeout after 60 seconds if nothing read
239                 auto timeout = serial::Timeout(1000U, 60000U, 0, 1000U, 0);
240                 std::unique_ptr<serial::Serial> serial = std::make_unique<serial::Serial>(port,
      baudrate, timeout);
241                 std::unique_ptr<RPLidar> lidar = std::make_unique<RPLidar>(port, baudrate,
      std::move(serial));
242                 return std::move(lidar);
243             }
244             catch (std::exception &e)
245             {
246                 return tl::make_unexpected(e.what());
247             }
248         }
249
256         RPLidar(const std::string &port, uint32_t baudrate, std::unique_ptr<serial::Serial> serial) :
      port(port), baudrate(baudrate), _serial(std::move(serial)){};
257
258         ~RPLidar() {}
259
260         void disconnect()
261         {
262             if (!this->_serial->isOpen())
263             {
264                 return;
265             }
266             this->_serial->close();
267         }
268
269         void set_motor_speed(int pwm)
270         {
271             if (0 <= pwm && pwm <= MAX_MOTOR_PWM)
272             {
273                 std::abort();
274             }
275             this->_motor_speed = pwm;
276             if (this->motor_running)
277                 this->_set_pwm(this->_motor_speed);
278         }
279
284         void start_motor()
285         {
286             spdlog::info("Starting motor");
287             // For A1
288             this->_serial->setDTR(false);
289
290             // For A2
291             this->_set_pwm(this->_motor_speed);
292             this->motor_running = true;
293         }
294
299         void stop_motor()
300         {
301             spdlog::info("Stopping motor");
```

```
302                 // For A2
303                 this->_set_pwm(0);
304                 std::this_thread::sleep_for(std::chrono::milliseconds(1));
305
306                 // For A1
307                 this->_serial->setDTR(true);
308                 this->motor_running = false;
309         }
310
316         tl::expected<DeviceInfo, std::string> get_info()
317         {
318                 // Check if there's data in the buffer
319                 if (this->_serial->available() > 0)
320                 {
321                     throw std::runtime_error("Data in buffer, you can't have info! Run flush() to empty the
      buffer.");
322                 }
323                 this->_send_cmd(GET_INFO_BYTE);
324
325                 uint8_t dsize;
326                 bool isSingle;
327                 uint8_t dtype;
328                 auto descriptor_result = this->_read_descriptor();
329                 if (!descriptor_result.has_value())
330                     return tl::make_unexpected(descriptor_result.error());
331                 std::tie(dsize, isSingle, dtype) = descriptor_result.value();
332
333                 // Check response properties
334                 if (dsize != INFO_LEN)
335                 {
336                     return tl::make_unexpected("Wrong get_info reply length");
337                 }
338                 if (!isSingle)
339                 {
340                     return tl::make_unexpected("Not a single response mode");
341                 }
342                 if (dtype != INFO_TYPE)
343                 {
344                     return tl::make_unexpected("Wrong response data type");
345                 }
346
347                 // Read the response
348                 std::vector<uint8_t> raw = this->_read_response(dsize);
349
350                 // Convert serial number to a hex string
351                 std::string serialNumber;
352                 for (size_t i = 4; i < raw.size(); ++i)
353                 {
354                     serialNumber += convertToHexString(raw[i]);
355                 }
356
357                 // Construct the device info struct
358                 DeviceInfo info;
359                 info.model = raw[0];
360                 info.firmware.first = raw[2];
361                 info.firmware.second = raw[1];
362                 info.hardware = raw[3];
363                 info.serialNumber = serialNumber;
364
365                 return info;
366         }
367
378         tl::expected<HealthInfo, std::string> get_health()
379         {
380                 // Check if there's data in the buffer
381                 if (this->_serial->available() > 0)
382                 {
383                     return tl::make_unexpected("Data in buffer, you can't get health info! Run cleanInput()
      to empty the buffer.");
384                 }
385                 spdlog::info("Asking for health");
386
387                 this->_send_cmd(GET_HEALTH_BYTE);
388
389                 // Read the descriptor
390                 uint8_t dsize;
391                 bool isSingle;
392                 uint8_t dtype;
393                 auto descriptor_result = this->_read_descriptor();
394                 if (!descriptor_result.has_value())
395                     return tl::make_unexpected(descriptor_result.error());
396                 std::tie(dsize, isSingle, dtype) = descriptor_result.value();
397
398                 // Check response properties
399                 if (dsize != HEALTH_LEN)
400                 {
401                     return tl::make_unexpected("Wrong get_health reply length");
```

```
402                    }
403                    if (!isSingle)
404                    {
405                        return tl::make_unexpected("Not a single response mode");
406                    }
407                    if (dtype != HEALTH_TYPE)
408                    {
409                        return tl::make_unexpected("Wrong response data type");
410                    }
411
412                    // Read the response
413                    std::vector<uint8_t> raw = this->_read_response(dsize);
414
415                    // Extract status and error code
416                    std::string status = HEALTH_STATUSES[raw[0]];
417                    int errorCode = (static_cast<int>(raw[1]) << 8) + static_cast<int>(raw[2]);
418
419                    return HealthInfo{status, errorCode};
420            }
421
426        void clean_input()
427        {
428            if (this->scanning.currently_scanning)
429            {
430                throw std::runtime_error("Cleaning not allowed during scanning process active!");
431            }
432            this->_serial->flushInput();
433            this->express_trame = 32;
434            this->express_data = nullptr;
435        }
436
442        void stop()
443        {
444            spdlog::info("Stopping scanning");
445            this->_send_cmd(STOP_BYTE);
446            std::this_thread::sleep_for(std::chrono::milliseconds(100));
447            this->scanning.currently_scanning = false;
448            this->clean_input();
449        }
450
456        tl::expected<nullptr_t, std::string> start(ScanType scanType = ScanType::NORMAL)
457        {
458            if (this->scanning.currently_scanning)
459            {
460                return tl::make_unexpected("Scanning already running!");
461            }
462
463            auto health_info_result = this->get_health();
464            if (!health_info_result.has_value())
465                return tl::make_unexpected(health_info_result.error());
466
467            HealthInfo healthInfo = health_info_result.value();
468            std::string status = healthInfo.status;
469            int errorCode = healthInfo.errorCode;
470            spdlog::debug("Health status: {} [{}}]", status, errorCode);
471
472            if (status == "Error")
473            {
474                this->reset();
475                health_info_result = this->get_health();
476                if (!health_info_result.has_value())
477                    return tl::make_unexpected(health_info_result.error());
478                healthInfo = health_info_result.value();
479                status = healthInfo.status;
480                errorCode = healthInfo.errorCode;
481                if (status == "Error")
482                {
483                    return tl::make_unexpected("RPLidar hardware failure. Error code: " +
     std::to_string(errorCode));
484                }
485            }
486            else if (status == "Warning")
487            {
488                spdlog::warn("Warning sensor status detected! Error code: {}", errorCode);
489            }
490
491            uint8_t cmd = SCAN_TYPE[scanType]["byte"];
492
493            spdlog::warn("starting scan process in {} mode", static_cast<int>(scanType));
494
495            if (scanType == ScanType::EXPRESS)
496            {
497                // Adjust this part according to your payload format
498                std::string payload = "\x00\x00\x00\x00\x00";
499                this->_send_payload_cmd(cmd, payload);
500            }
501            else
```

```
502                     {
503                         this->_send_cmd(cmd);
504                     }
505
506                     uint8_t dsize;
507                     bool isSingle;
508                     uint8_t dtype;
509
510                     auto descriptor_result = this->_read_descriptor();
511                     if (!descriptor_result.has_value())
512                         return tl::make_unexpected(descriptor_result.error());
513
514                     std::tie(dsize, isSingle, dtype) = descriptor_result.value();
515
516                     if (dsize != SCAN_TYPE[scanType]["size"])
517                     {
518                         return tl::make_unexpected("Wrong get_info reply length");
519                     }
520                     if (isSingle)
521                     {
522                         return tl::make_unexpected("Not a multiple response mode");
523                     }
524                     if (dtype != SCAN_TYPE[scanType]["response"])
525                     {
526                         return tl::make_unexpected("Wrong response data type");
527                     }
528                     this->scanning = ScanInfo{true, dsize, scanType};
529                     return nullptr;
530                 }
531
537             void reset()
538             {
539                 spdlog::info("Resetting the sensor");
540                 this->_send_cmd(RESET_BYTE);
541                 std::this_thread::sleep_for(std::chrono::milliseconds(2000));
542                 this->clean_input();
543             }
544
545             std::function<tl::expected<Measure, std::string>()> iter_measures(ScanType scanType =
         ScanType::NORMAL, int maxBufMeas = 3000)
546             {
547                 if (!this->motor_running)
548                 {
549                     this->start_motor();
550                 }
551
552                 if (!this->scanning.currently_scanning)
553                 {
554                     this->start(scanType);
555                 }
556
557                 // Define a lambda function to generate measures
558                 auto generator = [this, scanType, maxBufMeas]() -> tl::expected<Measure, std::string>
559                 {
560                     while (true)
561                     {
562                         int dsize = scanning.dsize;
563
564                         if (maxBufMeas != 0)
565                         {
566                             int dataInBuf = this->_serial->available();
567                             if (dataInBuf > maxBufMeas)
568                             {
569                                 spdlog::warn(
570                                     "Too many bytes in the input buffer: {}/{}. \n"
571                                     "Cleaning buffer...",
572                                     dataInBuf, maxBufMeas);
573                                 this->stop();
574                                 this->start(scanning.type);
575                             }
576                         }
577
578                         if (scanType == ScanType::NORMAL)
579                         {
580                             std::vector<uint8_t> raw = this->_read_response(dsize);
581                             return _process_scan(raw);
582                         }
583                         else if (scanType == ScanType::EXPRESS)
584                         {
585                             if (this->express_trame == 32)
586                             {
587                                 this->express_trame = 0;
588
589                                 if (this->express_data == nullptr)
590                                 {
591                                     spdlog::debug("reading first time bytes");
592                                     this->express_data =
```

```
            std::make_unique<ExpressPacket>(ExpressPacket(this->_read_response(dsize)));
593                         }
594
595                         this->express_old_data = std::move(this->express_data);
596                         spdlog::debug("set old_data with start_angle {}",
      this->express_old_data->start_angle);
597                         this->express_data =
      std::make_unique<ExpressPacket>(ExpressPacket(this->_read_response(dsize)));
598                         spdlog::debug("set new_data with start_angle {}",
      this->express_data->start_angle);
599                     }
600                     this->express_trame++;
601                     spdlog::debug("process scan of frame %d with angle : \n"
602                                   "%f and angle new : %f",
603                                   this->express_trame,
604                                   this->express_old_data->start_angle,
605                                   this->express_data->start_angle);
606                 Measure measure = _process_express_scan(this->express_old_data,
      this->express_data->start_angle, this->express_trame);
607                     return measure;
608                 }
609             }
610         };
611
612         return generator;
613     }
614
626     std::function<std::vector<Measure>()> iter_scans(ScanType scanType = ScanType::NORMAL, int
      maxBufMeas = 3000, int minLen = 5)
627     {
628         auto measureIterator = this->iter_measures(scanType, maxBufMeas);
629
630         // Define a lambda function to generate scans
631         auto scanGenerator = [measureIterator, minLen]() -> std::vector<Measure>
632         {
633             std::vector<Measure> scanList;
634             do
635             {
636                 tl::expected<Measure, std::string> maybe_measure = measureIterator();
637                 if (!maybe_measure.has_value())
638                 {
639                     continue;
640                 }
641                 Measure measure = maybe_measure.value();
642                 bool newScan = measure.newScan;
643                 if (newScan)
644                 {
645                     if (scanList.size() > minLen)
646                     {
647                         break;
648                     }
649                     scanList.clear();
650                 }
651                 if (measure.distance > 0)
652                 {
653                     scanList.push_back(measure);
654                 }
655             }
656             while (true);
657             return scanList;
658         };
659         return scanGenerator;
660     }
661
662  private:
663     void _set_pwm(int pwm)
664     {
665         std::string payload;
666         payload.push_back(static_cast<uint8_t>(pwm & 0xFF));
667         payload.push_back(static_cast<uint8_t>((pwm >> 8) & 0xFF));
668         this->_send_payload_cmd(SET_PWM_BYTE, payload);
669     }
670
677     void _send_payload_cmd(uint8_t cmd, const std::string &payload)
678     {
679         // Calculate the size
680         uint8_t size = static_cast<uint8_t>(payload.size());
681
682         // Construct the request string
683         std::string req;
684         req += static_cast<uint8_t>(SYNC_BYTE);
685         req += static_cast<uint8_t>(cmd);
686         req += static_cast<uint8_t>(size);
687         req += payload;
688
689         // Calculate the checksum
690         uint8_t checksum = 0;
```

```
691                 for (const uint8_t &c : req)
692                 {
693                     checksum ^= static_cast<uint8_t>(c);
694                 }
695
696                 req += static_cast<uint8_t>(checksum);
697
698                 this->_serial->write(req);
699                 spdlog::debug("Command sent: {}", spdlog::to_hex(req));
700             }
701
707         void _send_cmd(uint8_t cmd)
708         {
709             std::string req;
710             req += static_cast<uint8_t>(SYNC_BYTE);
711             req += static_cast<uint8_t>(cmd);
712
713             this->_serial->write(req);
714             spdlog::debug("Command sent: {}", spdlog::to_hex(req));
715         }
716
722         tl::expected<
723             std::tuple<uint8_t, bool, uint8_t>,
724             std::string>
725         _read_descriptor()
726         {
727             // Read descriptor packet
728             std::array<uint8_t, DESCRIPTOR_LEN> descriptor;
729             memset(descriptor.data(), '\0', sizeof(descriptor));
730             this->_serial->read(descriptor.data(), DESCRIPTOR_LEN);
731             spdlog::debug("Received descriptor: {}", spdlog::to_hex(descriptor));
732
733             if (descriptor.size() != DESCRIPTOR_LEN)
734             {
735                 return tl::make_unexpected("Descriptor length mismatch");
736             }
737             else if (descriptor[0] != SYNC_BYTE || descriptor[1] != SYNC_BYTE2)
738             {
739                 return tl::make_unexpected("Incorrect descriptor starting bytes");
740             }
741
742             bool isSingle = descriptor[5] == 0;
743             return std::make_tuple(descriptor[2], isSingle, descriptor[6]);
744         }
745
752         std::vector<uint8_t> _read_response(int dsize)
753         {
754             spdlog::debug("Trying to read response: {} bytes", dsize);
755
756             std::vector<uint8_t> data;
757             data.reserve(dsize);
758
759             while (this->_serial->available() < dsize)
760             {
761                 std::this_thread::sleep_for(std::chrono::milliseconds(1));
762             }
763
764             this->_serial->read(data, dsize);
765
766             spdlog::debug("Received data: {}", spdlog::to_hex(data));
767             return data;
768         }
769
770         std::string convertToHexString(uint8_t value)
771         {
772             // Convert a uint8_t to a hexadecimal string
773             std::stringstream stream;
774             stream << std::hex << std::uppercase << std::setw(2) << std::setfill('0') <<
    static_cast<int>(value);
775             return stream.str();
776         }
777
778         tl::expected<Measure, std::string> _process_scan(const std::vector<uint8_t> &raw)
779         {
780             Measure measurementData;
781
782             bool newScan = static_cast<bool>(raw[0] & 0b1);
783             bool inversedNewScan = static_cast<bool>((raw[0] >> 1) & 0b1);
784             int quality = static_cast<int>(raw[0] >> 2);
785
786             if (newScan == inversedNewScan)
787             {
788                 return tl::make_unexpected("New scan flags mismatch");
789             }
790
791             int checkBit = static_cast<int>(raw[1] & 0b1);
792             if (checkBit != 1)
```

```
793                {
794                    return tl::make_unexpected("Check bit not equal to 1");
795                }
796
797                int anglePart1 = static_cast<int>(raw[1] >> 1);
798                int anglePart2 = static_cast<int>(raw[2]) << 7;
799                float angle = (anglePart1 + anglePart2) / 64.0;
800
801                int distancePart1 = static_cast<int>(raw[3]);
802                int distancePart2 = static_cast<int>(raw[4]) << 8;
803                float distance = (distancePart1 + distancePart2) / 4.0;
804
805                measurementData.newScan = newScan;
806                measurementData.quality = quality;
807                measurementData.angle = angle;
808                measurementData.distance = distance;
809
810                return measurementData;
811            }
812
824            Measure _process_express_scan(std::unique_ptr<ExpressPacket> &data, float newAngle, int trame)
825            {
826                Measure measurementData;
827
828                bool newScan = (newAngle < data->start_angle) && (trame == 1);
829
830                float angle = std::fmod((data->start_angle + ((newAngle - data->start_angle) / 32 * trame -
     data->angle[trame - 1])), 360);
831                float distance = data->distance[trame - 1];
832
833                measurementData.newScan = newScan;
834                measurementData.quality = 0; // Replace this with your actual quality value
835                measurementData.angle = angle;
836                measurementData.distance = distance;
837
838                return measurementData;
839            }
840
841    private:
842        std::unique_ptr<serial::Serial> _serial = nullptr;
843
844        std::string port;
845        uint32_t baudrate;
846
847        int _motor_speed = DEFAULT_MOTOR_PWM;
848        bool motor_running = false;
849        ScanInfo scanning = ScanInfo{false, 0, ScanType::NORMAL};
850        int express_trame = 32;
851
852        std::unique_ptr<ExpressPacket> express_data = nullptr;
853        std::unique_ptr<ExpressPacket> express_old_data = nullptr;
854    };
855 }
856
857 #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.
```cpp
1 // From: https://stackoverflow.com/a/59522794
2
3 #include <array>
4 #include <cstddef>
5
6 namespace utils
7 {
8     template <typename T>
9     constexpr const auto &RawTypeName()
10     {
11 #ifdef _MSC_VER
12         return __FUNCSIG__;
13 #else
14         return __PRETTY_FUNCTION__;
15 #endif
16     }
17
18     struct RawTypeNameFormat
19     {
20         std::size_t leading_junk = 0, trailing_junk = 0;
21     };
22
23     // Returns 'false' on failure.
24     inline constexpr bool GetRawTypeNameFormat(RawTypeNameFormat *format)
25     {
26         const auto &str = RawTypeName<int>();
27         for (std::size_t i = 0;; i++)
28         {
29             if (str[i] == 'i' && str[i + 1] == 'n' && str[i + 2] == 't')
30             {
31                 if (format)
32                 {
33                     format->leading_junk = i;
34                     format->trailing_junk = sizeof(str) - i - 3 - 1; // '3' is the length of "int", '1'
    is the space for the null terminator.
35                 }
36                 return true;
37             }
38         }
39         return false;
40     }
41
42     inline static constexpr RawTypeNameFormat format =
43         []
44     {
45         static_assert(GetRawTypeNameFormat(nullptr), "Unable to figure out how to generate type names on
    this compiler.");
46         RawTypeNameFormat format;
47         GetRawTypeNameFormat(&format);
48         return format;
49     }();
50
51     // Returns the type name in a 'std::array<char, N>' (null-terminated).
52     template <typename T>
53     [[nodiscard]] constexpr auto CexprTypeName()
54     {
55         constexpr std::size_t len = sizeof(RawTypeName<T>()) - format.leading_junk -
    format.trailing_junk;
56         std::array<char, len> name{};
57         for (std::size_t i = 0; i < len - 1; i++)
58             name[i] = RawTypeName<T>()[i + format.leading_junk];
```

```
59          return name;
60      }
61
62      template <typename T>
63      [[nodiscard]] const char *TypeName()
64      {
65          static constexpr auto name = CexprTypeName<T>();
66          return name.data();
67      }
68      template <typename T>
69      [[nodiscard]] const char *TypeName(const T &)
70      {
71          return TypeName<T>();
72      }
73 }
```

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.
```
1 #ifndef UTILITY_HPP
2 #define UTILITY_HPP
3
4 #pragma once
5
6 #include <string>
7 #include <sstream>
8 #include <vector>
9
10 namespace utils {
11     // https://stackoverflow.com/a/46931770
12     static std::vector<std::string> split(const std::string& s, const int from_index, const int to_index,
    char delim) {
13         std::vector<std::string> result;
14         std::stringstream ss(s);
15         std::string item;
16         int index = 0;
17
18         while (getline(ss, item, delim)) {
19             if (index >= from_index && index < to_index)
20                 result.push_back(item);
21             index++;
22         }
23
24         return result;
25     }
26
27     // https://stackoverflow.com/a/46711735
28     static constexpr uint32_t hash(const std::string_view s) noexcept
29     {
30         uint32_t hash = 5381;
31
32         for (const char* c = s.data(); c < s.data() + s.size(); ++c)
33             hash = ((hash << 5) + hash) + (unsigned char)*c;
34
35         return hash;
36     }
37
38     // From: https://stackoverflow.com/a/5665377
39     // To escape HTML characters
```

```
40    static void encode(std::string& data) {
41        std::string buffer;
42        buffer.reserve(data.size());
43        for (size_t pos = 0; pos != data.size(); ++pos) {
44            switch (data[pos]) {
45            case '&':  buffer.append("&amp;");       break;
46            case '\"': buffer.append("&quot;");      break;
47            case '\'': buffer.append("&apos;");      break;
48            case '<':  buffer.append("&lt;");        break;
49            case '>':  buffer.append("&gt;");        break;
50            default:   buffer.append(&data[pos], 1); break;
51            }
52        }
53        data.swap(buffer);
54    }
55
56    // To get class name without namespace
57    static std::string getStringAfterLastColon(const std::string& input) {
58        size_t last_colon_pos = input.find_last_of("::");
59
60        if (last_colon_pos != std::string::npos) {
61            return input.substr(last_colon_pos + 1);
62        }
63        else {
64            return input;
65        }
66    }
67 }
68
69 #endif
```

Index