Version: 4.0.1

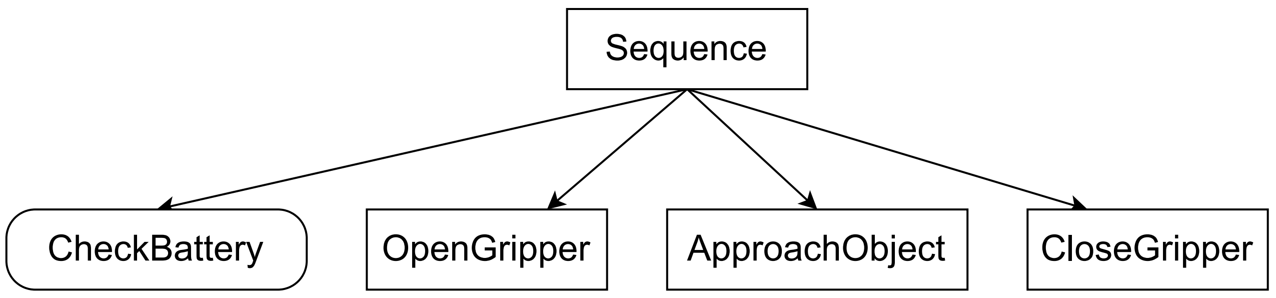
Your first Behavior Tree

Behavior Trees, similar to State Machines, are nothing more than a mechanism to invoke **callbacks** at the right time under the right conditions. What happens inside these callbacks is up to you.

We will use the expression **"to invoke the callback"** and **"to tick"** interchangeably.

In this tutorial series, most of the time our dummy Actions will simply print some information on the console, but keep in mind that real "production" code would probably do something more complicated.

Further, we will create this simple tree:



How to create your own ActionNodes

The default (and recommended) way to create a TreeNode is by inheritance.

```
// Example of custom SyncActionNode (synchronous action)
// without ports.
class ApproachObject : public BT::SyncActionNode
{
public:
  ApproachObject(const std::string& name) :
      BT::SyncActionNode(name, {})
  {}
  // You must override the virtual function tick()
  BT::NodeStatus tick() override
  {
    std::cout << "ApproachObject: " << this->name() << std::endl;
    return BT::NodeStatus::SUCCESS;
  }
};
```

As you can see:

- Any instance of a TreeNode has a `name`. This identifier is meant to be human-readable and it **doesn't** need to be unique.
- The method **tick()** is the place where the actual Action takes place. It must always return a `NodeStatus`, i.e. RUNNING, SUCCESS or FAILURE.

Alternatively, we can use **dependency injection** to create a TreeNode given a function pointer (i.e. "functor").

The only requirement of the functor is to have either one of these signatures:

```
BT::NodeStatus myFunction()
BT::NodeStatus myFunction(BT::TreeNode& self)
```

For example:

```
using namespace BT;

// Simple function that return a NodeStatus
BT::NodeStatus CheckBattery()
{
  std::cout << "[ Battery: OK ]" << std::endl;
  return BT::NodeStatus::SUCCESS;
}

// We want to wrap into an ActionNode the methods open() and close()
class GripperInterface
{
public:
  GripperInterface(): _open(true) {}

  NodeStatus open()
```

```
    {
      _open = true;
      std::cout << "GripperInterface::open" << std::endl;
      return NodeStatus::SUCCESS;
    }

    NodeStatus close()
    {
      std::cout << "GripperInterface::close" << std::endl;
      _open = false;
      return NodeStatus::SUCCESS;
    }

  private:
    bool _open; // shared information
  };
```

We can build a `SimpleActionNode` from any of these functors:

- CheckBattery()
- GripperInterface::open()
- GripperInterface::close()

Create a tree dynamically with an XML

Let's consider the following XML file named **my_tree.xml**:

```xml
 <root BTCPP_format="4" >
     <BehaviorTree ID="MainTree">
        <Sequence name="root_sequence">
            <CheckBattery   name="check_battery"/>
            <OpenGripper    name="open_gripper"/>
            <ApproachObject name="approach_object"/>
            <CloseGripper   name="close_gripper"/>
        </Sequence>
     </BehaviorTree>
 </root>
```

TIP

You can find more details about the XML schema here.

We must first register our custom TreeNodes into the `BehaviorTreeFactory` and then load the XML from file or text.

The identifier used in the XML must coincide with those used to register the TreeNodes.

The attribute "name" represents the name of the instance; **it is optional**.

```
#include "behaviortree_cpp/bt_factory.h"

// file that contains the custom nodes definitions
#include "dummy_nodes.h"
using namespace DummyNodes;

int main()
{
    // We use the BehaviorTreeFactory to register our custom nodes
  BehaviorTreeFactory factory;

  // The recommended way to create a Node is through inheritance.
  factory.registerNodeType<ApproachObject>("ApproachObject");

  // Registering a SimpleActionNode using a function pointer.
  // Here we prefer to use a lambda,but you can use std::bind too
  factory.registerSimpleCondition("CheckBattery", [&](){ return
CheckBattery(); });

  // You can also create SimpleActionNodes using methods of a class.
  GripperInterface gripper;
  factory.registerSimpleAction("OpenGripper", [&](){ return gripper.open(); }
);
  factory.registerSimpleAction("CloseGripper", [&](){ return gripper.close();
}

  // Trees are created at deployment-time (i.e. at run-time, but only
  // once at the beginning).

  // IMPORTANT: when the object "tree" goes out of scope, all the
  // TreeNodes are destroyed
   auto tree = factory.createTreeFromFile("./my_tree.xml");

  // To "execute" a Tree you need to "tick" it.
  // The tick is propagated to the children based on the logic of the tree.
  // In this case, the entire sequence is executed, because all the children
  // of the Sequence return SUCCESS.
  tree.tickWhileRunning();

  return 0;
}

/* Expected output:
*
  [ Battery: OK ]
  GripperInterface::open
  ApproachObject: approach_object
  GripperInterface::close
*/
```

Edit this page