Version: 4.0.1

Ports with generic types

In the previous tutorials we introduced input and output ports, where the type of the port was a `std::string`.

Next, we will show how to assign generic C++ types to your ports.

Parsing a string

BehaviorTree.CPP supports the automatic conversion of strings into common types, such as `int`, `long`, `double`, `bool`, `NodeStatus`, etc. User-defined types can be supported easily as well.

For instance:

```
// We want to use this custom type
struct Position2D
{
  double x;
  double y;
};
```

To allow the XML loader to instantiate a `Position2D` from a string, we need to provide a template specialization of `BT::convertFromString<Position2D>(StringView)`.

It is up to you how `Position2D` is serialized into a string; in this case, we simply separate two numbers with a *semicolon*.

```
// Template specialization to converts a string to Position2D.
namespace BT
{
    template <> inline Position2D convertFromString(StringView str)
```

```
    {
        // We expect real numbers separated by semicolons
        auto parts = splitString(str, ';');
        if (parts.size() != 2)
        {
            throw RuntimeError("invalid input)");
        }
        else{
            Position2D output;
            output.x     = convertFromString<double>(parts[0]);
            output.y     = convertFromString<double>(parts[1]);
            return output;
        }
    }
} // end namespace BT
```

- `StringView` is a C++11 version of std::string_view. You can pass either a `std::string` or a `const char*`.
- The library provides a simple `splitString` function. Feel free to use another one, like boost::algorithm::split.
- We can use the specialization `convertFromString<double>()`.

Example

As we did in the previous tutorial, we can create two custom Actions, one will write into a port and the other will read from a port.

```
class CalculateGoal: public SyncActionNode
{
  public:
    CalculateGoal(const std::string& name, const NodeConfig& config):
      SyncActionNode(name,config)
    {}

    static PortsList providedPorts()
    {
      return { OutputPort<Position2D>("goal") };
    }

    NodeStatus tick() override
    {
```

```
      Position2D mygoal = {1.1, 2.3};
      setOutput<Position2D>("goal", mygoal);
      return NodeStatus::SUCCESS;
    }
};

class PrintTarget: public SyncActionNode
{
  public:
    PrintTarget(const std::string& name, const NodeConfig& config):
        SyncActionNode(name,config)
    {}

    static PortsList providedPorts()
    {
      // Optionally, a port can have a human readable description
      const char*  description = "Simply print the goal on console...";
      return { InputPort<Position2D>("target", description) };
    }

    NodeStatus tick() override
    {
      auto res = getInput<Position2D>("target");
      if( !res )
      {
        throw RuntimeError("error reading port [target]:", res.error());
      }
      Position2D target = res.value();
      printf("Target positions: [ %.1f, %.1f ]\n", target.x, target.y );
      return NodeStatus::SUCCESS;
    }
};
```

We can now connect input/output ports as usual, pointing to the same entry of the Blackboard.

The tree in the next example is a Sequence of 4 actions

- Store a value of `Position2D` in the entry **GoalPosition**, using the action `CalculateGoal`.
- Call `PrintTarget`. The input "target" will be read from the Blackboard entry **GoalPosition**.

- Use the built-in action `Script` to assign the string "-1;3" to the key **OtherGoal**. The conversion from string to `Position2D` will be done automatically.

- Call `PrintTarget` again. The input "target" will be read from the entry **OtherGoal**.

```
static const char* xml_text = R"(

 <root BTCPP_format="4" >
     <BehaviorTree ID="MainTree">
        <Sequence name="root">
            <CalculateGoal goal="{GoalPosition}" />
            <PrintTarget   target="{GoalPosition}" />
            <Script        code=" OtherGoal:='-1;3' " />
            <PrintTarget   target="{OtherGoal}" />
        </Sequence>
     </BehaviorTree>
 </root>
 )";

int main()
{
  BT::BehaviorTreeFactory factory;
  factory.registerNodeType<CalculateGoal>("CalculateGoal");
  factory.registerNodeType<PrintTarget>("PrintTarget");

  auto tree = factory.createTreeFromText(xml_text);
  tree.tickWhileRunning();

  return 0;
}
/* Expected output:

    Target positions: [ 1.1, 2.3 ]
    Converting string: "-1;3"
    Target positions: [ -1.0, 3.0 ]
*/
```

Edit this page