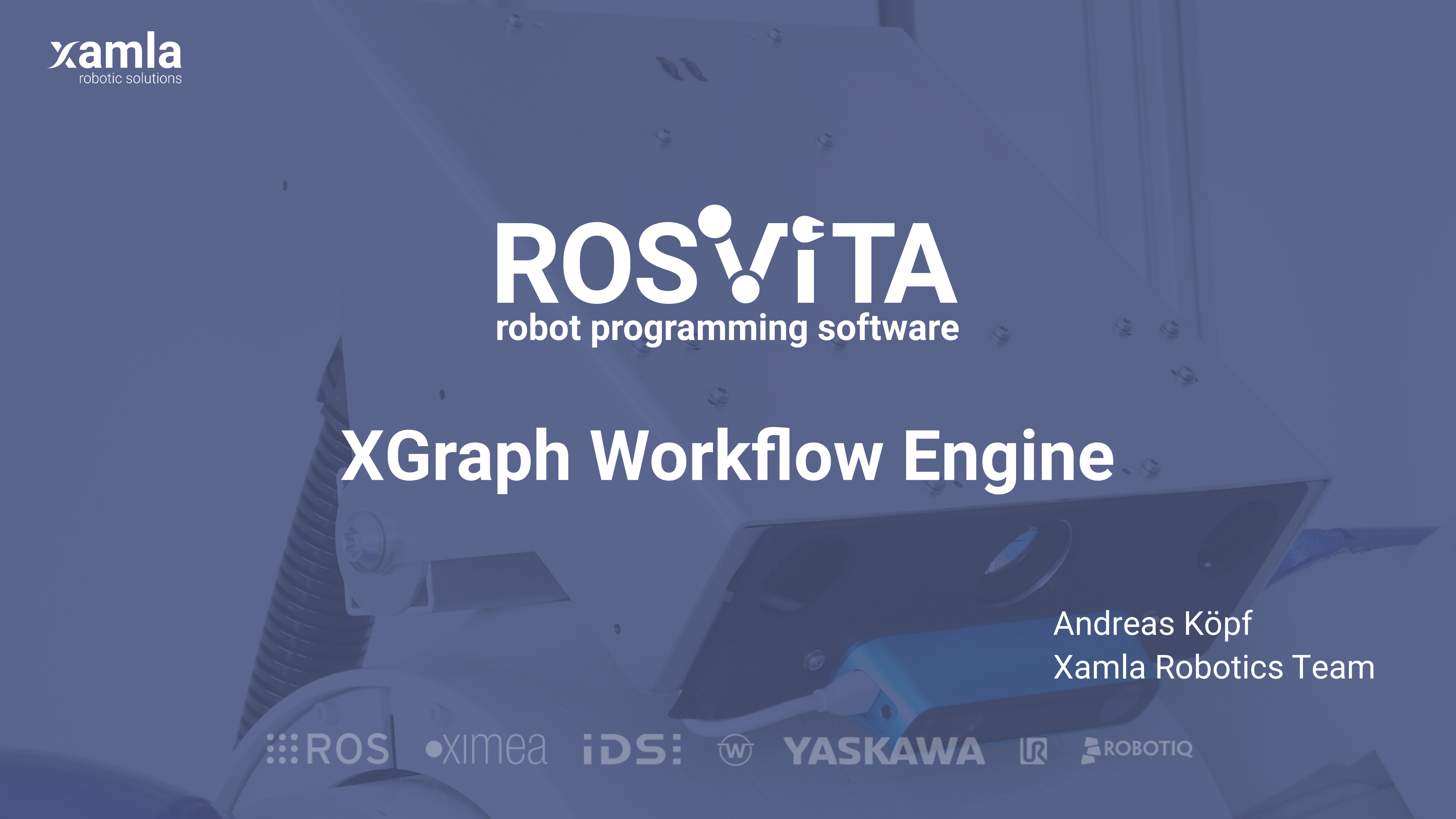
xamla
robotic solutions



ROSViTA
robot programming software

XGraph Workflow Engine

Andreas Köpf
Xamla Robotics Team

ROS ximea iDS YASKAWA ROBOTIQ

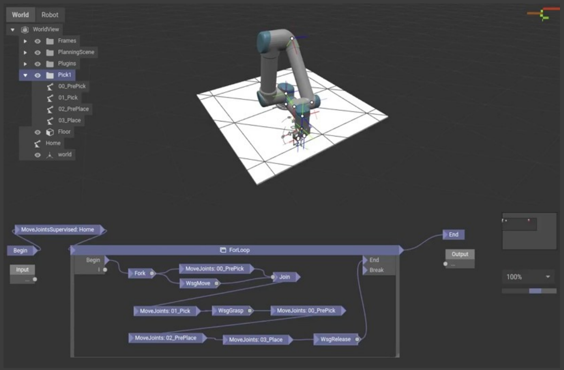
Talk Structure

1. What is XGraph?
2. Basic Concepts
3. Video Walk-Through
4. Scripting
5. .Net Extensibility
6. Converter Details
7. Execution Semantics
8. Outlook

1 _ What is XGraph?

xamla
robotic solutions

The Rosvita XGraph Workflow Engine

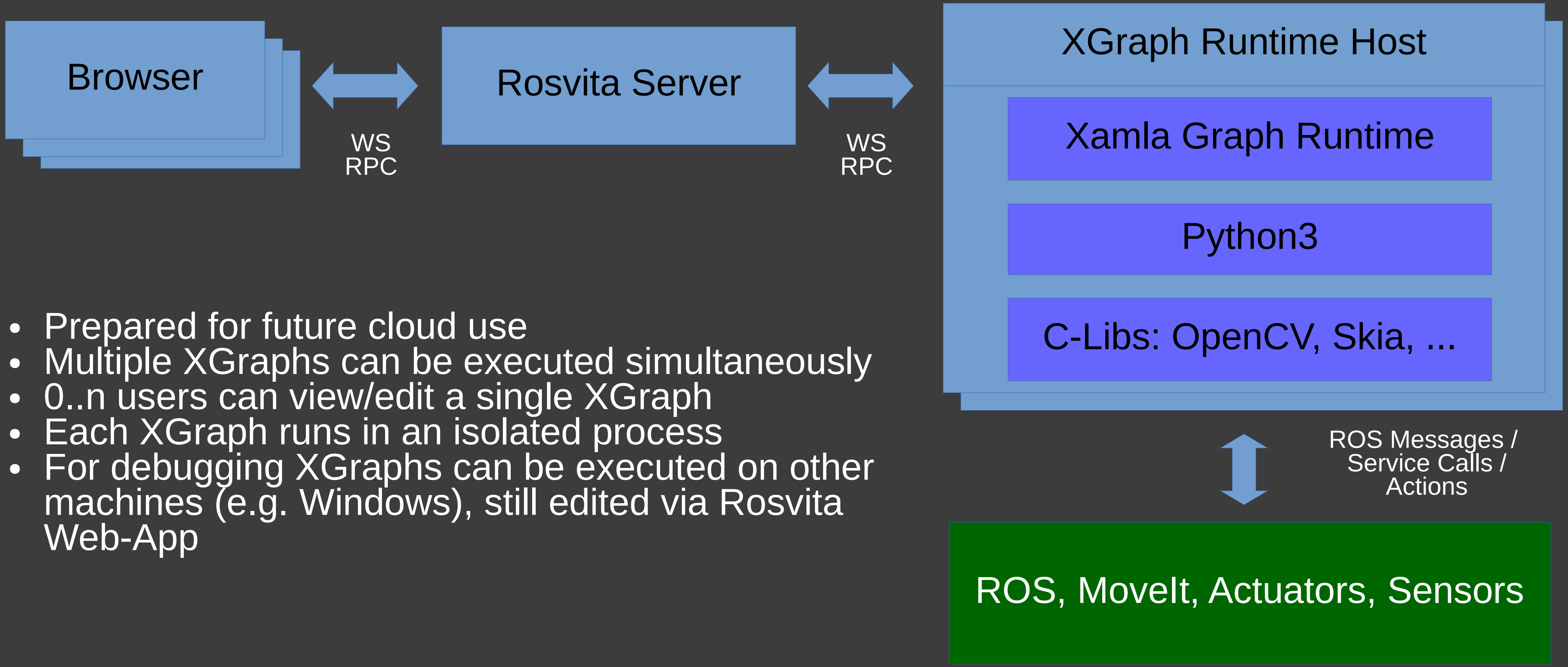


A graphical programming system for Rosvita

Primary use-cases:
Robot operations, sensor input & generic data processing

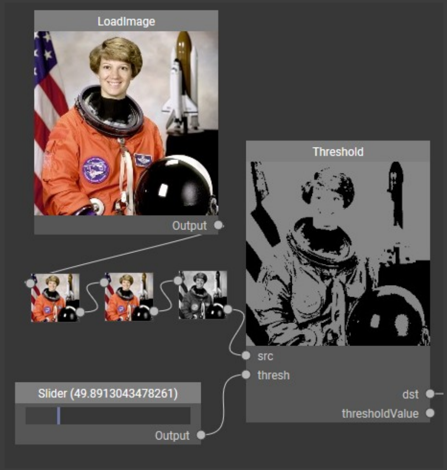
- Strictly typed Visual Programming Language running on .Net Core
- Executable graphical representation of programs
- Usable by non-programmers (to some extend)
- Allows prototyping, explorative development & live parameter adjustments
- Easily extensible via .Net (C#) and Python scripts

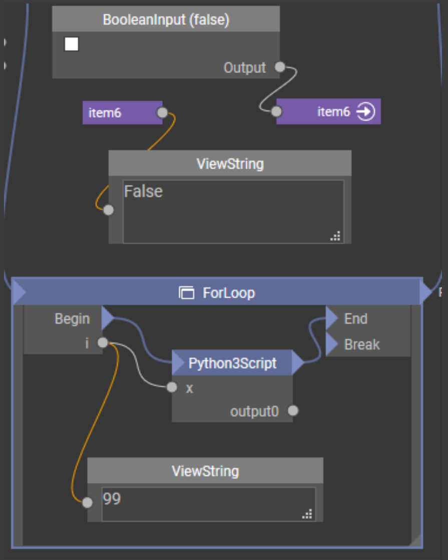
Architectural Overview


Browser
WS RPC
Rosvita Server
WS RPC
XGraph Runtime Host
Xamla Graph Runtime
Python3
C-Libs: OpenCV, Skia, ...
ROS Messages / Service Calls / Actions
ROS, MoveIt, Actuators, Sensors

- Prepared for future cloud use
- Multiple XGraphs can be executed simultaneously
- 0..n users can view/edit a single XGraph
- Each XGraph runs in an isolated process
- For debugging XGraphs can be executed on other machines (e.g. Windows), still edited via Rosvita Web-App

2 _ Basic Concepts



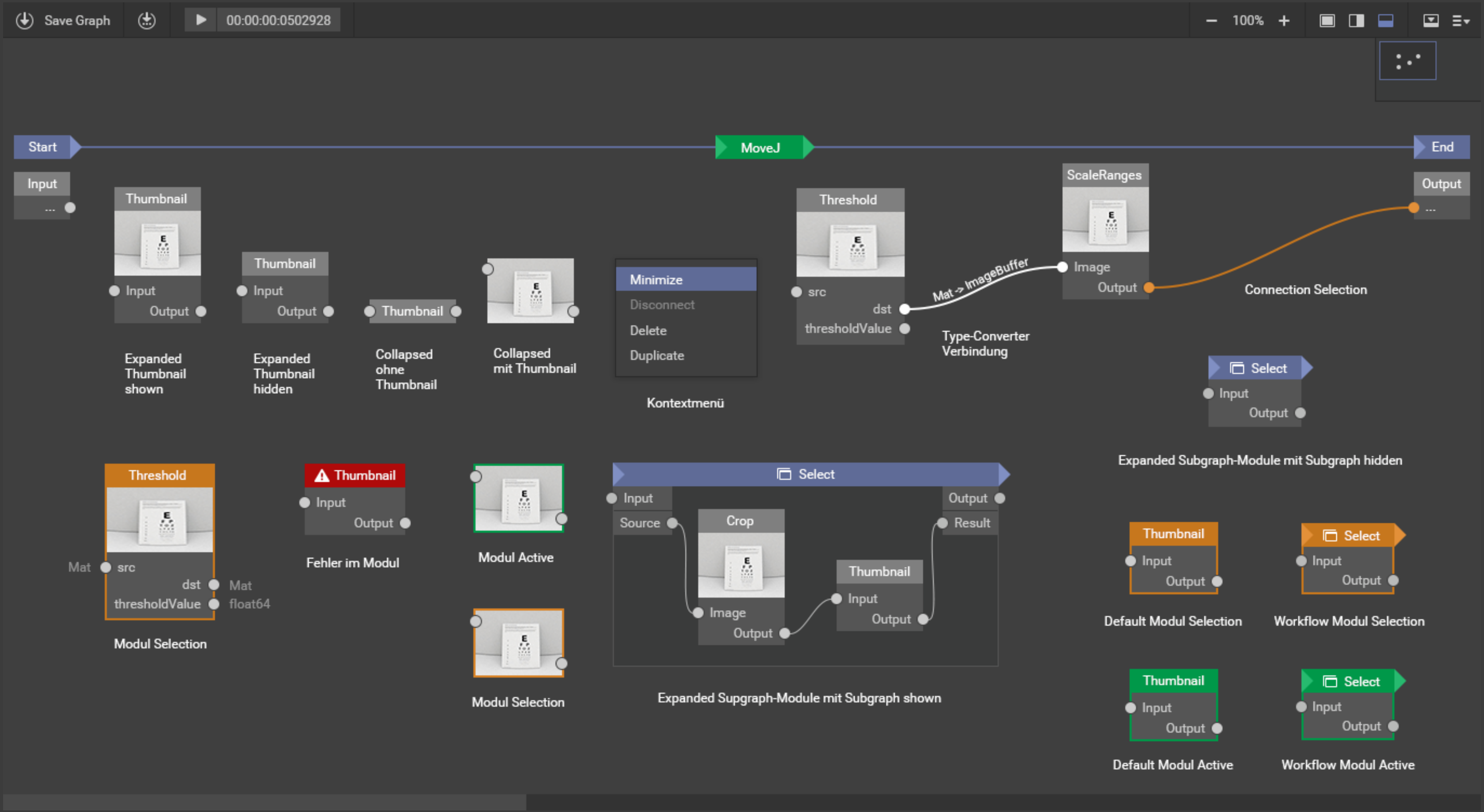


Elements of an XGraph

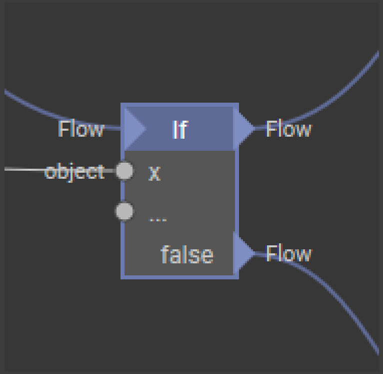
- Graph canvas
- Modules
- Pins
- Cables: Value, Converters, Flow
- Property Editor
- Interface Modules (Input & Output, Begin & End)
- Sub-Graph Modules (Select, SelectMany)
- Graph-Instances
- Control-Modules (text box, checkbox, slider)
- Code & Script Modules (Python, C#)
- Comments
- Ports (hidden long range connections)

In development: Value inspection system

Basic Concepts



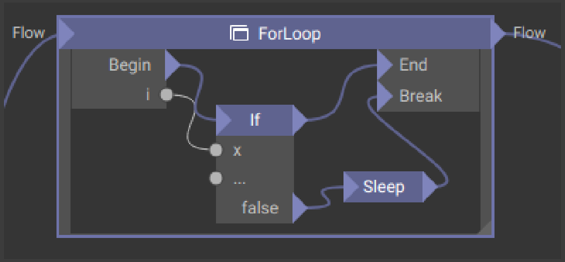
Concept: Flow



Why?

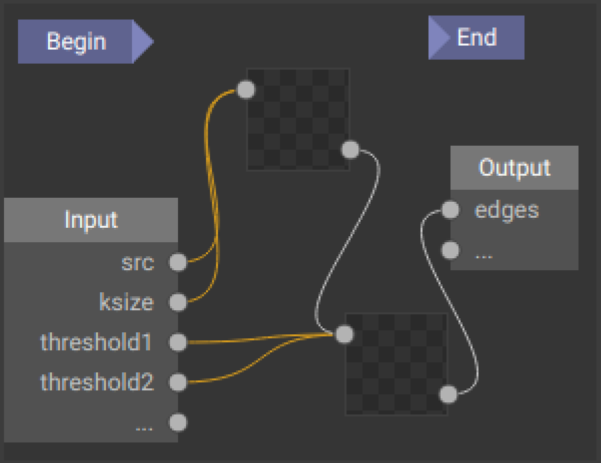
- Side-effects of robot operations require strict ordering
- Provide conditional branching and controlled parallel execution options

- Blue cabels are flow connections



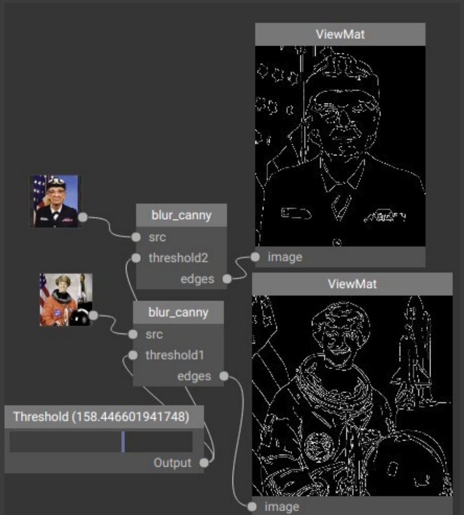
- Flow modules:
- For-Loop, For-Each
- If
- Fork/Join
- Throw/Catch (in development)

Concept: Graph Instances



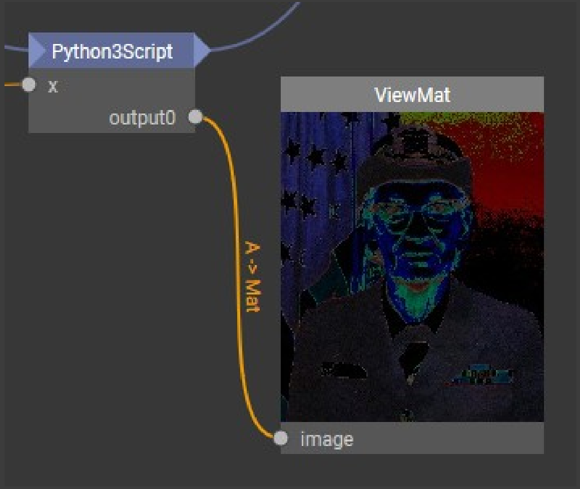
Why?

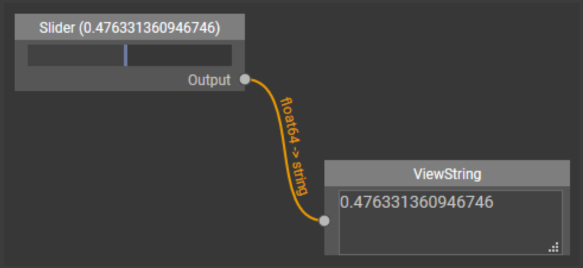
- Composition: Break down larger graphs into chunks
- Reuse: Instantiate one graph in different workflows



- To create a graph-instance: Drag a .xgraph file from file explorer onto a graph canvas
- A relative path is stored to a graph instance source graph
- If Begin & End are not connected a graph becomes pure functional (the graph instance has no flow)
- Methodology: Develop each reusable graph together with a minimal test graph that instantiates it and can be used for debugging

Concept: Implicit Value Conversion



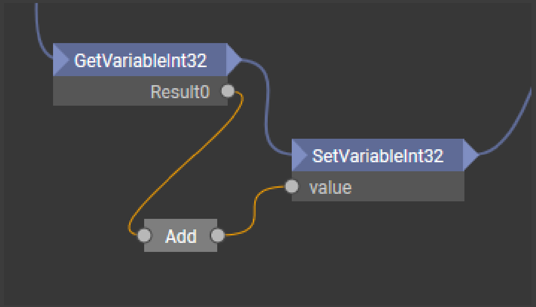


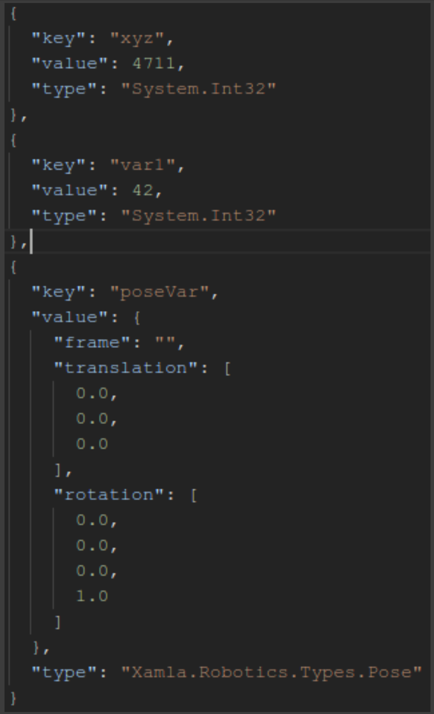
Why?

- In a strictly typed system often types do not match (e.g. int16, int32, int64, float32, float64)
- Different libraries use different classes for the same data (e.g. OpenCV Mat vs. Skia Bitmap)

Orange Cables indicate a value conversion.

- Conversion to intermediate Xamla types (library neutral) is used to effectively allow N:M library conversions
- Even implicit conversion between push & pull is possible (e.g. IEnumerable ↔ IObservable)





```
{
  "key": "xyz",
  "value": 4711,
  "type": "System.Int32"
},
{
  "key": "var1",
  "value": 42,
  "type": "System.Int32"
},
{
  "key": "poseVar",
  "value": {
    "frame": "",
    "translation": [
      0.0,
      0.0,
      0.0
    ],
    "rotation": [
      0.0,
      0.0,
      0.0,
      1.0
    ]
  },
  "type": "Xamla.Robotics.Types.Pose"
}
```

Concept: Persistent Value Store

Why?

- Store variables over multiple graph executions
- Provide settings from outside via JSON file (.xgraph.store.json)

Modules to access the Value Store:

- GetVariable, SetVariable (object with default)
- Typed: GetVariabe{Boolean, Int32, Float64, String}
- Typed: SetVariabe{Boolean, Int32, Float64, String}
- RemoveVariable, ClearStore
- SaveStore

3 _ Video Walk-Through

XGraph Tutorial Videos

XGraph Walkthrough

- ✓ **Rosvita XGraph Walkthrough**
  https://youtu.be/LgHPmnLkLqI

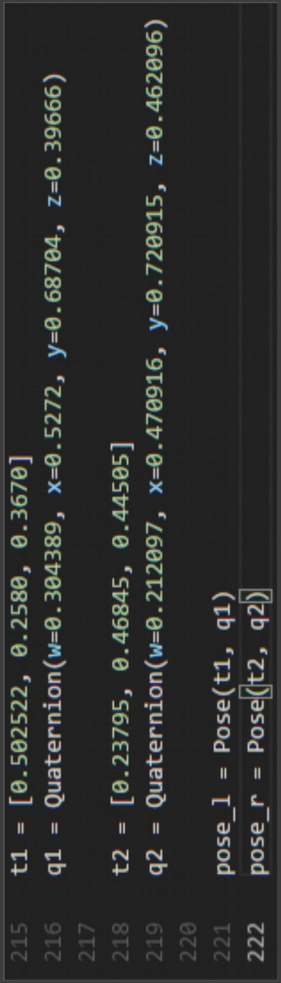
Pick and Place

- ✓ **Pick and Place Workflow**
  https://youtu.be/MJAHPZibfrA

Camera Modules

- ✓ **Rosvita Camera Modules**
  https://youtu.be/P9CzGynAyBM

4 _ Scripting



```
t1 = [0.502522, 0.2580, 0.3670]
q1 = Quaternion(w=0.304389, x=0.5272, y=0.68704, z=0.39666)

t2 = [0.23795, 0.46845, 0.44505]
q2 = Quaternion(w=0.212097, x=0.470916, y=0.720915, z=0.462096)

pose_l = Pose(t1, q1)
pose_r = Pose(t2, q2)
```

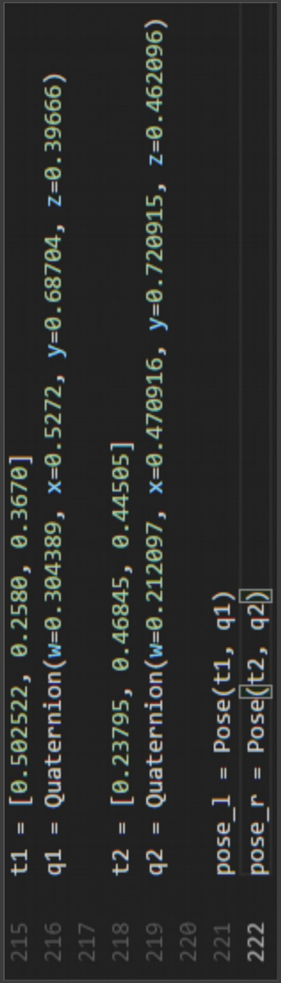
Script Modules

If a module is missing powerful scripting options are available to fill the gap:

- C#   CSharp.Code, CSharpScript Modules
- Python3   Python3Eval, Python3Script, Python3ScriptFile

→ PythonEval & CSharpScript are for simple expressions

- Python Libs can be placed in <projet>/libs/python folder
- Use xamla_motion for Python Robot interactio: https://github.com/Xamla/pythonClientLib_XamlaMotion
- Recommended: Develop & test Python scripts externally and then paste / reference them inside an XGraph



```
t1 = [0.502522, 0.2580, 0.3670]
q1 = Quaternion(w=0.304389, x=0.5272, y=0.68704, z=0.39666)

t2 = [0.23795, 0.46845, 0.44505]
q2 = Quaternion(w=0.212097, x=0.470916, y=0.720915, z=0.462096)

pose_l = Pose(t1, q1)
pose_r = Pose(t2, q2)
```

Python Script Module Details

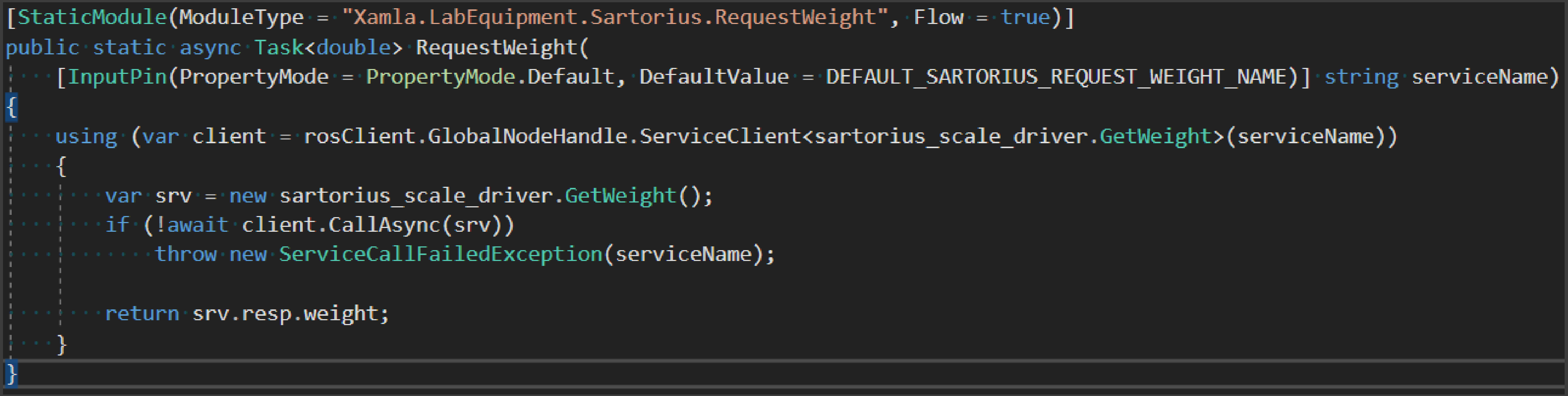
- The module signatue is generated using Python3 type hints placed in the Python function signature. Arguments → Pins

- Signature is automatically updated after edits (e.g. new, renamed, deleted pins)
- Primitives, List, Dict and ndarray are automatically converted between Python and .Net types
- Use Tuple for multiple return values
- Numpy, Scikit, OpenCV etc. can be used inside Python modules
- Nvidia-docker for GPU use (e.g. DeepLearning) available soon

5 _ .Net Extensibility

Writing Custom Modules in .Net

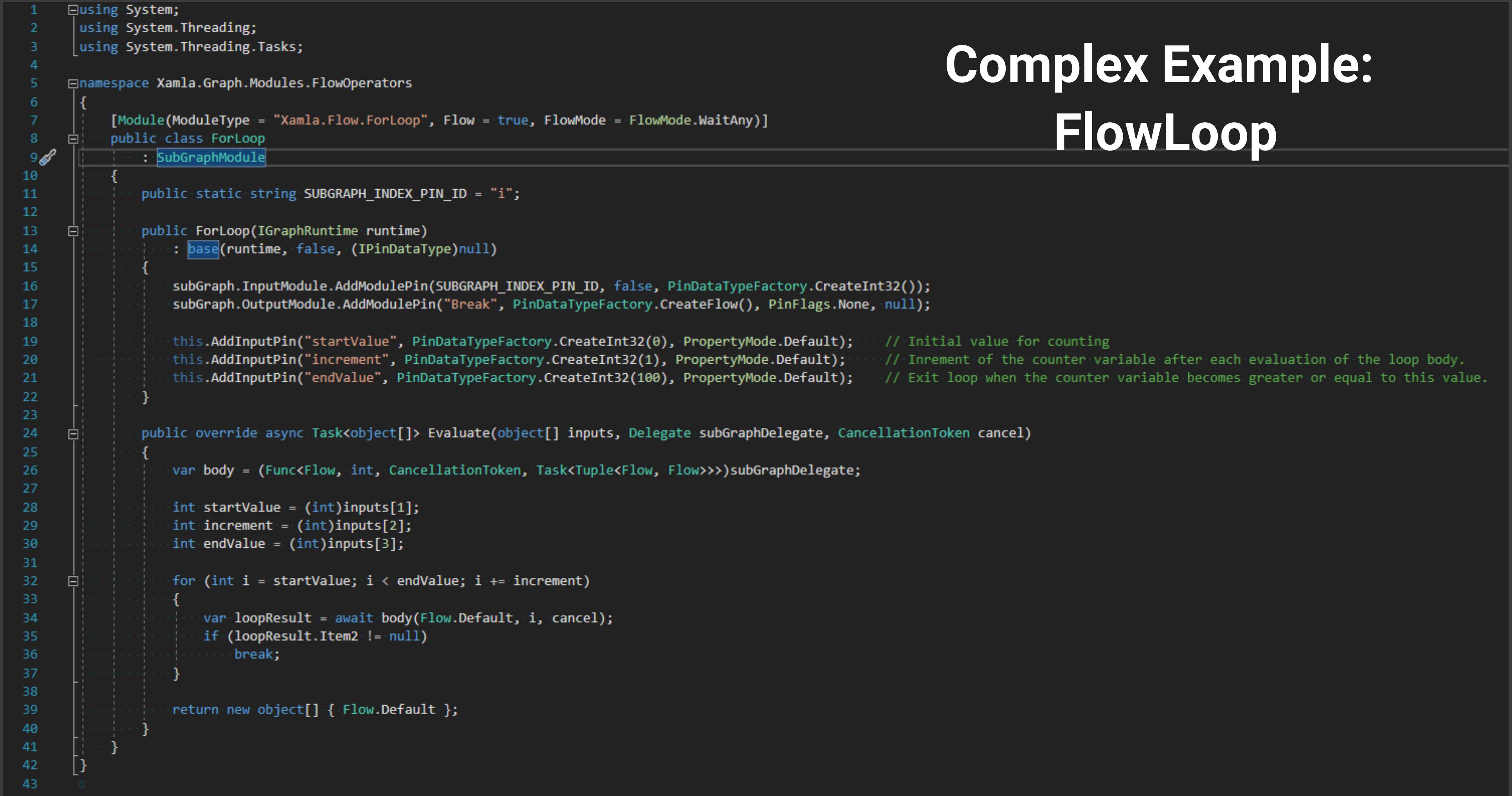
The simple way: StaticMethod Module via Attributes



```
[StaticModule(ModuleType = "Xamla.LabEquipment.Sartorius.RequestWeight", Flow = true)]
public static async Task<double> RequestWeight(
    [InputPin(PropertyMode = PropertyMode.Default, DefaultValue = DEFAULT_SARTORIUS_REQUEST_WEIGHT_NAME)] string serviceName)
{
    using (var client = rosClient.GlobalNodeHandle.ServiceClient<sartorius_scale_driver.GetWeight>(serviceName))
    {
        var srv = new sartorius_scale_driver.GetWeight();
        if (!await client.CallAsync(srv))
            throw new ServiceCallFailedException(serviceName);

        return srv.resp.weight;
    }
}
```

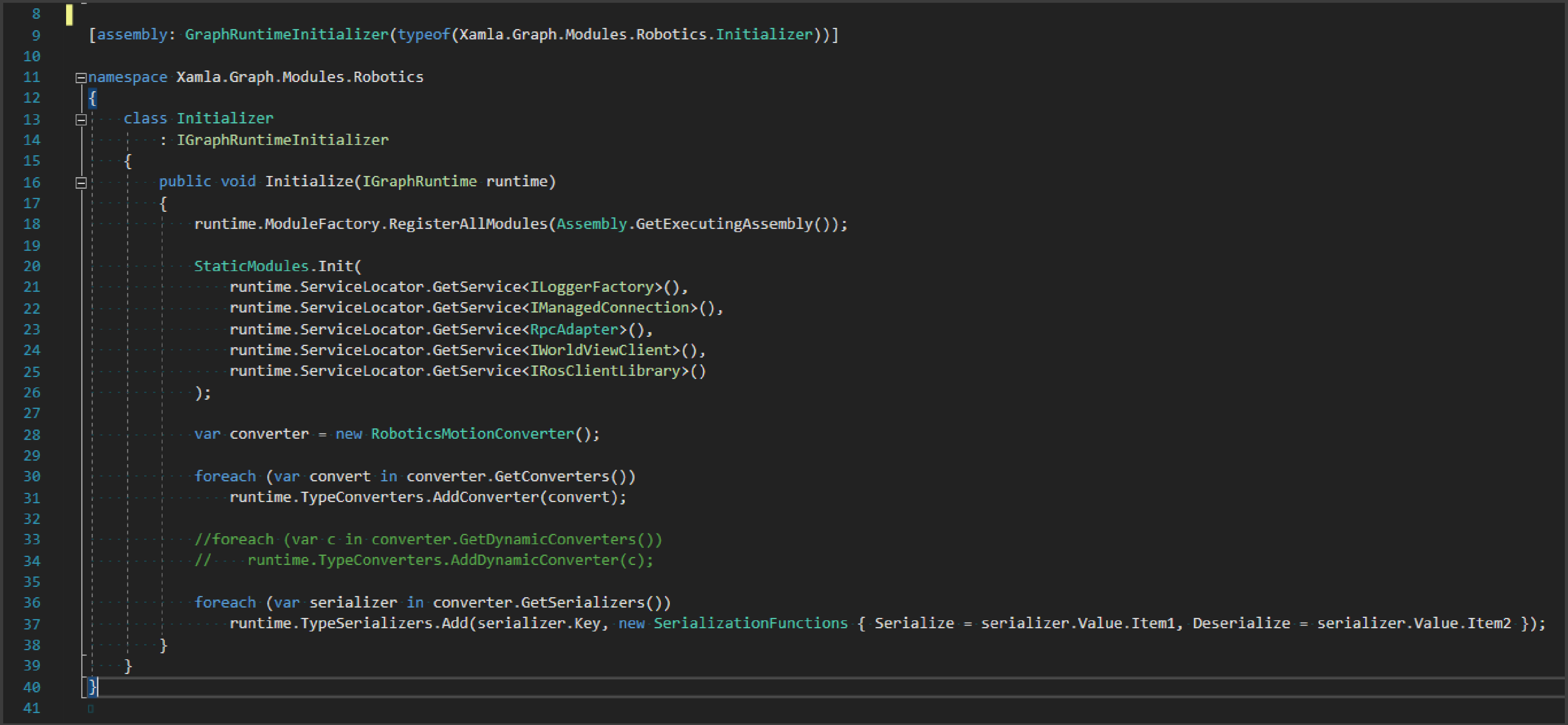
.Net Extensibility

Complex Example: FlowLoop



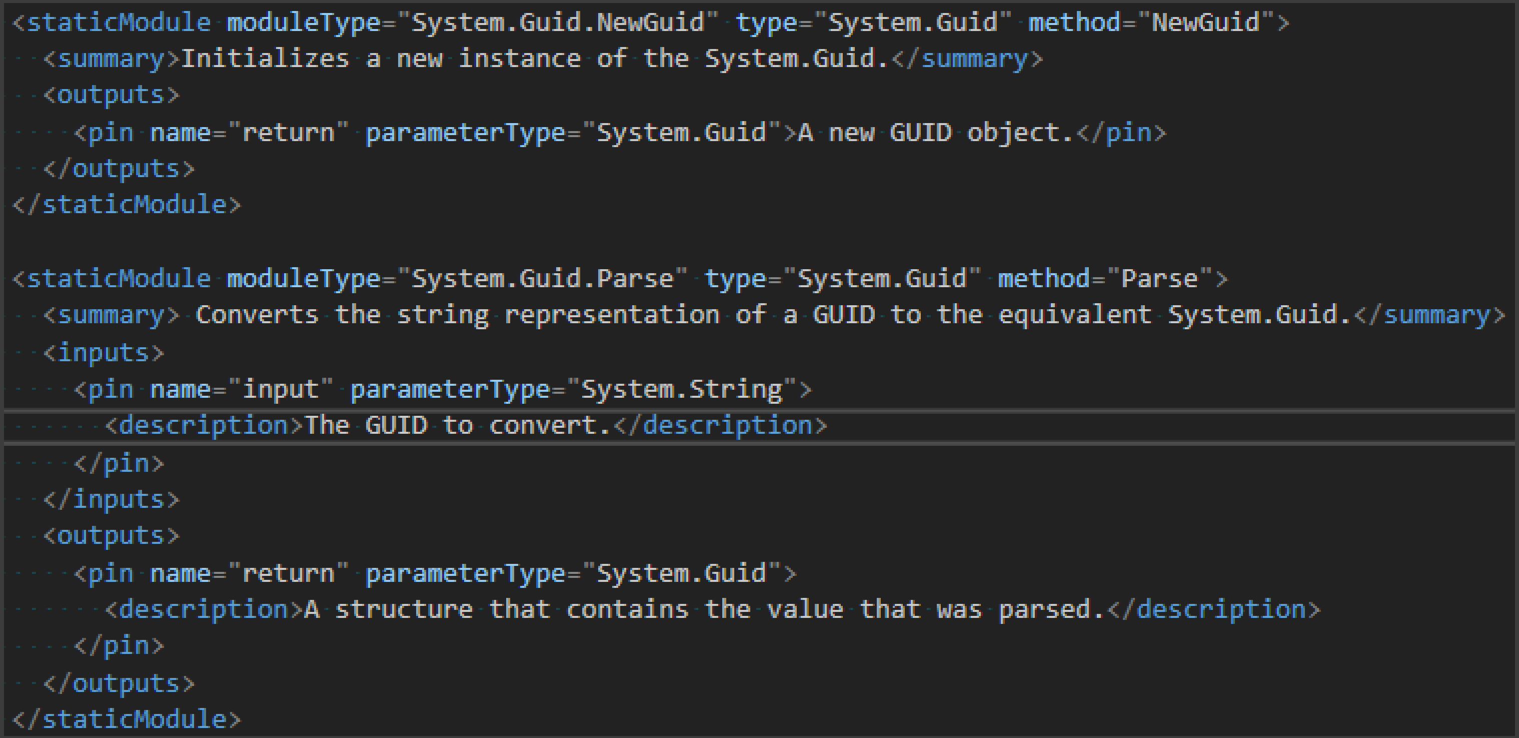
```csharp
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Xamla.Graph.Modules.FlowOperators
{
    [Module(ModuleType = "Xamla.Flow.ForLoop", Flow = true, FlowMode = FlowMode.WaitAny)]
    public class ForLoop
        : SubGraphModule
    {
        public static string SUBGRAPH_INDEX_PIN_ID = "i";

        public ForLoop(IGraphRuntime runtime)
            : base(runtime, false, (IPinDataType)null)
        {
            subGraph.InputModule.AddModulePin(SUBGRAPH_INDEX_PIN_ID, false, PinDataTypeFactory.CreateInt32());
            subGraph.OutputModule.AddModulePin("Break", PinDataTypeFactory.CreateFlow(), PinFlags.None, null);

            this.AddInputPin("startValue", PinDataTypeFactory.CreateInt32(0), PropertyMode.Default);    // Initial value for counting
            this.AddInputPin("increment", PinDataTypeFactory.CreateInt32(1), PropertyMode.Default);     // Inrement of the counter variable after each evaluation of the loop body.
            this.AddInputPin("endValue", PinDataTypeFactory.CreateInt32(100), PropertyMode.Default);    // Exit loop when the counter variable becomes greater or equal to this value.
        }

        public override async Task<object[]> Evaluate(object[] inputs, Delegate subGraphDelegate, CancellationToken cancel)
        {
            var body = (Func<Flow, int, CancellationToken, Task<Tuple<Flow, Flow>>>)subGraphDelegate;

            int startValue = (int)inputs[1];
            int increment = (int)inputs[2];
            int endValue = (int)inputs[3];

            for (int i = startValue; i < endValue; i += increment)
            {
                var loopResult = await body(Flow.Default, i, cancel);
                if (loopResult.Item2 != null)
                    break;
            }

            return new object[] { Flow.Default };
        }
    }
}
```

Initializers & Dependency Injection



```
[assembly: GraphRuntimeInitializer(typeof(Xamla.Graph.Modules.Robotics.Initializer))]

namespace Xamla.Graph.Modules.Robotics
{
    class Initializer
        : IGraphRuntimeInitializer
    {
        public void Initialize(IGraphRuntime runtime)
        {
            runtime.ModuleFactory.RegisterAllModules(Assembly.GetExecutingAssembly());

            StaticModules.Init(
                runtime.ServiceLocator.GetService<ILoggerFactory>(),
                runtime.ServiceLocator.GetService<IManagedConnection>(),
                runtime.ServiceLocator.GetService<RpcAdapter>(),
                runtime.ServiceLocator.GetService<IWorldViewClient>(),
                runtime.ServiceLocator.GetService<IRosClientLibrary>()
            );

            var converter = new RoboticsMotionConverter();

            foreach (var convert in converter.GetConverters())
                runtime.TypeConverters.AddConverter(convert);

            //foreach (var c in converter.GetDynamicConverters())
            //      runtime.TypeConverters.AddDynamicConverter(c);

            foreach (var serializer in converter.GetSerializers())
                runtime.TypeSerializers.Add(serializer.Key, new SerializationFunctions { Serialize = serializer.Value.Item1, Deserialize = serializer.Value.Item2 });
        }
    }
}
```

.xmodule Files



```xml
<staticModule moduleType="System.Guid.NewGuid" type="System.Guid" method="NewGuid">
  <summary>Initializes a new instance of the System.Guid.</summary>
  <outputs>
    <pin name="return" parameterType="System.Guid">A new GUID object.</pin>
  </outputs>
</staticModule>

<staticModule moduleType="System.Guid.Parse" type="System.Guid" method="Parse">
  <summary> Converts the string representation of a GUID to the equivalent System.Guid.</summary>
  <inputs>
    <pin name="input" parameterType="System.String">
      <description>The GUID to convert.</description>
    </pin>
  </inputs>
  <outputs>
    <pin name="return" parameterType="System.Guid">
      <description>A structure that contains the value that was parsed.</description>
    </pin>
  </outputs>
</staticModule>
```

- Contain XML descriptions to convert modules from static .Net functions without attributes
- Empty .xmodule files, act as sentinal file for graph module assembly discovery (allows drag&drop deployment of module assemblies)

6 _ Converter Details

Converter Details: Intermediate Types

Intermedita Data Types help to avoid a quadratic number of library-to-library converters.

- A Multi-dimensional array
- V 1d vector
- M 2d matrix
- I ImageBuffer (2d multi-channel)

- Examples:
  OpenCv.Mat → I → Skia.Bitmap
  ImageBuffer → A → np.ndarray

Converter Pitfalls

- Information might be lost, e.g. the A type does not carry information about image channels, therefore BGR might become RGB
- The type converter system tries to use all kinds of base classes to find an intermediate type:
  Select orange connections to see if it makes sense for you!
- Sometimes unintuitive conversions are selected, e.g. if a string is converted into a sequence a single element sequence of string is created (not a sequence of characters)

7 _ Execution Semantics

Semantics of an XGraph [1/2]

- Lazy: Only connected modules are evaluated
- Modules without output pins have invisible sink connections: e.g. WriteFile, ViewImage
- Flow carries Execute-Signal or Exception
- Non-Flow modules are evaluated (in parallel) as soon as values for their inputs have been generated
- Fow modules additionally wait to receive a flow signal
- Flow is cancelled when first flow-signal reaches End module
- Join: Currently wait completes on 1st exception: Fail-early
- One Evaluation Context per Graph / Sub-graph canvas
- Non-flow modules are evaluated only once per evaluation of their context

Semantics of an XGraph [2/2]

- Pin connections: Other modules or Property Container
- Pin cardinalities (default):
  - Value: In: 1 Out: n
  - Flow: In: n Out: 1
- Input pins in Sub-Graphs can be connected to outputs of modules in outer graphs (but not the other way round)
- Generic modules compute pin type & type converter upon connection
- Sequences are lazy-evaluated (infinite generators)
- Sequences soures can be interactive or reactive (e.g. events)

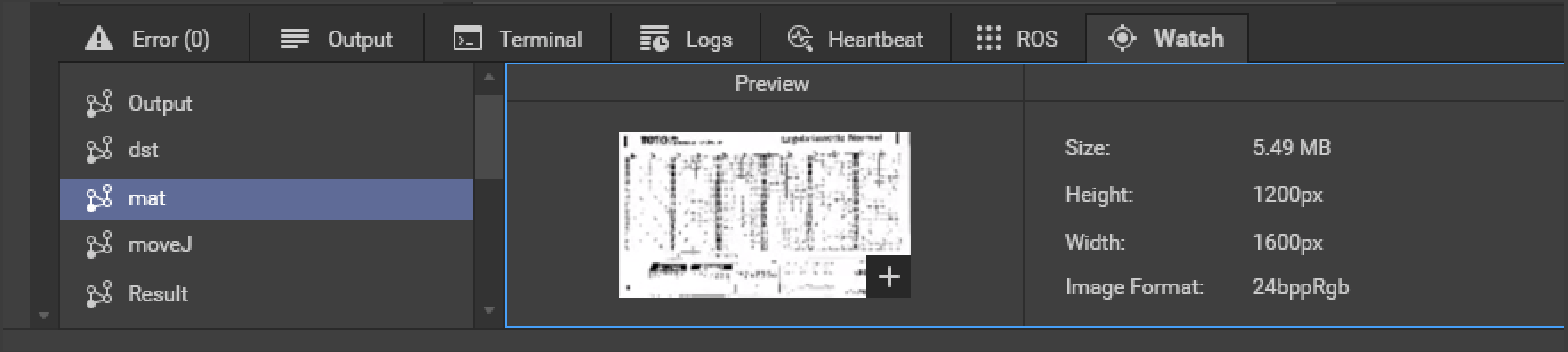
8 _ Outlook

Future Development

1000+ ideas exist - prioritization is the hard thing!
Here are some near-term candidates:

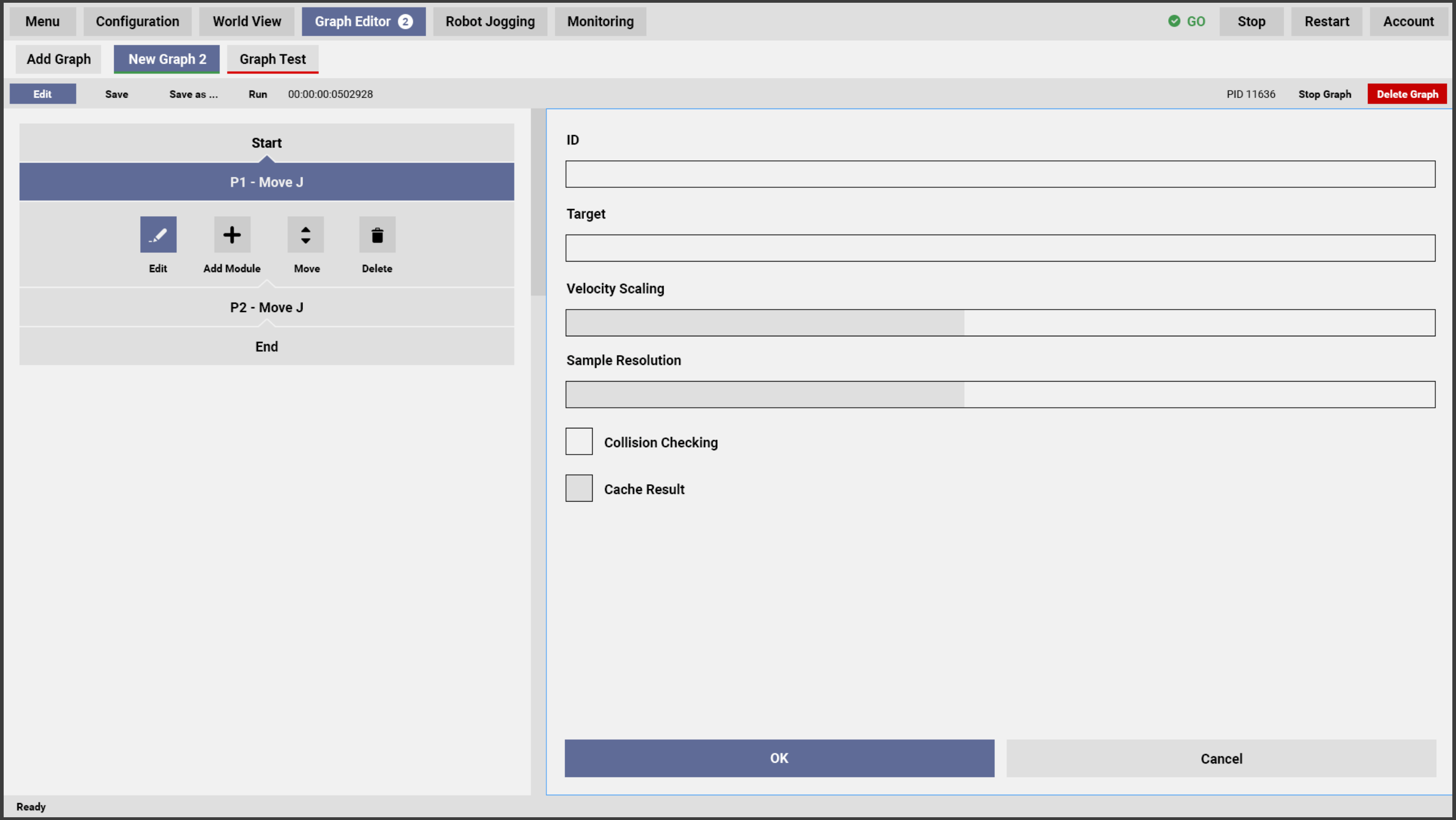
- Improve error display & handling
- Inspection/Output Visualizer system
- Flow-Module Stacking (simplified display)
- More Drag&Drop Options, e.g. JointValues, Poses, Paths, Trajectories
- Provide option lists in Property Editor (e.g. available MoveGroups, Action names for Grippers etc.)
- Simplify Navigation to Script Source-Files (double click)
- Sub-Graph extraction via range-selection
- XML Copy & Paste module sharing (via Mail, Chat etc.)
- .Net ↔ Python Robotic Type conversions
- Python module registry
- Online module registry
- Undo system
- Option to pause a graph
- Improve robustness of library calls (e.g. OpenCV)
- Re-evaluation of non-flow modules due to flow source changes
- Generic ROS Modules
- Auto-Start Graph
- Value Plotting Modules
- 3D Processing Modules (e.g. Point Clouds)

Inspection System / Watch Window



Size:	5.49 MB
Height:	1200px
Width:	1600px
Image Format:	24bppRgb

Joystick friendly Waypoint Teaching



ID

Target

Velocity Scaling

Sample Resolution

□ Collision Checking

□ Cache Result

xamla
robotic solutions

Thank you for your attention!

Questions?

Tutorial Videos:
http://www.youtube.com/xamla