ifm electronic

IoTCore.Net Programming Guide

Version 2.0

CONTENTS

1 THE IOTCORE.NET API

1.1 TARGET PLATFORM

The target platforms for the IoTCore.Net are all platforms, which support the .NET Standard 2.0 specification.

1.2 CREATE AN IOTCORE

The first thing you need to do is to create a new instance of the IoTCore class by calling the “IoTCoreFactory.Create()” method. The method takes a name as parameter and passes it to the constructor of the IoTCore class. The constructor of the IoTCore class internally creates a device element member, which serves as the root element for the IoTCore element tree. The device element is accessible via the “Root” member of the IoTCore object. The constructor of the IoTCore class passes this name to the root device element as its identifier. You can access the current version of the IoTCore via the “Version” member of the IoTCore object.

```
namespace Sample01
{
    using System;
    using ifmIoTCore;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");
                Console.WriteLine(ioTCore.Version);
                Console.WriteLine(ioTCore.Root.Identifier);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
}
```

1.3 ADD ELEMENTS TO THE IOT-CORE ELEMENT TREE

The IoTCore supports elements of type device, structure, service, event, and data. The IoTCore organizes the elements in a hierarchical tree. Each element has an identifier, which is a mandatory field. The path of the element in the IoT-Core element tree together with its identifier form the element address, which uniquely identifies the element. Additionally, each element can have a format specification, a list of profiles to which it belongs, and an additional unique identifier. Each element also provides a generic member field for user data. This field gives you the possibility to reference a user specific object from the element. To add an element to the element tree, call the corresponding create element function from the IoTCore element manager. The create element functions have an additional parameter, that defines whether the “treechanged” event of the parent element is raised or not.

1.3.1 Add a structure element

A structure element serves as a container in which elements that semantically belong together are organized.

To add a new structure element to the IoTCore, create an object of the class “StructureElement” and add it to the IoTCore element tree.

```
namespace Sample02
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
}
```

1.3.2 Add a service element

A service element represents an executable service that you can invoke internally or by a remote service request.

The service element class constructor takes the service handler function in its constructor. The service handler function can receive incoming data and return outgoing data. The data type of the object that is passed into the service handler function or returned from it is specified in the template parameter of the service element. The service handler receives a reference to the service element from which it is called and the optional context id of the service call. For different service handler input and output parameters there are different service element classes available.

The “ActionServiceElement” class defines a service handler function that receives no input data and returns no output data. The “ReaderServiceElement” class defines a service handler function that has no incoming data but returns outgoing data. The “WriterServiceElement” class defines a service handler function that has incoming data but returns no outgoing data. The “ServiceElement” class defines a service handler function that has incoming data and returns outgoing data.

To add a new service element to the IoTCore, create an object of one of the service element classes and add it to the IoTCore element tree.

```
namespace Sample03
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;

    internal class UserData : IEquatable<UserData>
    {
        public int Int1;
        public float Float1;
        public string String1;
```

```
        public UserData()
        {
            Int1 = 10;
            Float1 = 1.2345f;
            String1 = "Hallo";
        }

        public bool Equals(UserData other)
        {
            if (other == null) return false;
            return Int1 == other.Int1 &&
                   Float1.Equals(other.Float1) &&
                   String1 == other.String1;
        }

        public override string ToString()
        {
            return $"Int1={Int1} Float1={Float1} String1={String1}";
        }
    }

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root, "struct1", null,
                    new List<string> { "profile1" });

                ioTCore.CreateActionServiceElement(struct1,
                    "service1",
                    HandleService1);

                ioTCore.CreateGetterServiceElement<int>(struct1,
                    "service2",
                    HandleService2);

                ioTCore.CreateSetterServiceElement<string>(struct1,
                    "service3",
                    HandleService3);

                ioTCore.CreateServiceElement<int, string>(struct1,
                    "service4",
                    HandleService4);

                ioTCore.CreateServiceElement<UserData, UserData>(struct1,
                    "service5",
                    HandleService5);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static void HandleService1(IBaseElement sender, int? cid)
        {
            Console.WriteLine("Do something");
        }

        private static int HandleService2(IBaseElement sender, int? cid)
        {
            Console.WriteLine("Do something and return result");
            return 0;
        }

        private static void HandleService3(IBaseElement sender, string data, int? cid)
        {
            Console.WriteLine($"Do something with {data}");
        }

        private static string HandleService4(IBaseElement sender, int data, int? cid)
```

```
        {
            Console.WriteLine($"Do something with {data} and return result");
            return $"Received {data}";
        }

        private static UserData HandleService5(IBaseElement sender, UserData data, int? cid)
        {
            Console.WriteLine($"Do something with {data} and return result");
            return data;
        }
    }
}
```

1.3.3 Add an event element

An event element represents an event.

You can subscribe to an event element to receive notifications when the event fires. It provides the service elements “subscribe” and “unsubscribe”.

To add a new event element to the IoTCore, create an object of the class “EventElement” and add it to the IoTCore element tree.

```
namespace Sample04
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                ioTCore.CreateEventElement(struct1, "event1");
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
}
```

1.3.4 Add a data element

A data element represents a data point.

The “DataElement” create function takes the service handler functions for the getter and setter methods in its constructor, as well as a flag that specifies whether a “datachanged” event element is created or not. The service handler functions are provided by the application. If the setter service handler function return “true” the data element will raise a “datachanged” event; otherwise not. The service handler functions receive a reference to the data element from which it is called.

The data type of the object that the data element represents is specified as a template parameter of the data element class.

The data element can provide different format specifiers for the different types of data objects it represents. Supported data types are bool, integer, float, string, integer enumeration, string enumeration, array, and object.

Data elements support data caching. To enable data caching, pass a cache timeout value in the constructor of the class. If the timeout value is null, data caching is disabled.

1.3.4.1 Add a bool data element

To add a new bool data element to the IoTCore, create an object of the class “DataElement” with a bool format specifier and add it to the IoTCore element tree.

```
namespace Sample05
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                ioTCore.CreateDataElement(struct1,
                    "bool1",
                    GetBool1,
                    SetBool1,
                    true,
                    true,
                    true,
                    format: new BooleanFormat());
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static bool GetBool1(IBaseElement sender)
        {
            return _bool1;
        }

        private static bool SetBool1(IBaseElement sender, bool value)
        {
            if (_bool1 == value) return false;
            _bool1 = value;
            return true;
        }
        private static bool _bool1 = true;
    }
}
```

1.3.4.2 Add an integer data element

To add a new integer data element to the IoTCore, create an object of the class “DataElement” with an integer format specifier and add it to the IoTCore element tree.

```
namespace Sample06
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.Valuations;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                ioTCore.CreateDataElement(struct1,
                    "int1",
                    GetInt1,
                    SetInt1,
                    true,
                    true,
                    true,
                    format: new IntegerFormat(new IntegerValuation(0, 100)));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static int GetInt1(IBaseElement sender)
        {
            return _int1;
        }

        private static bool SetInt1(IBaseElement sender, int value)
        {
            if (_int1 == value) return false;
            _int1 = value;
            return true;
        }
        private static int _int1 = 10;
    }
}
```

1.3.4.3 Add a float data element

To add a new float data element to the IoTCore, create an object of the class “DataElement” with a float format specifier and add it to the IoTCore element tree.

```
namespace Sample07
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.Valuations;
    using ifmIoTCore.Utilities;

    internal class Program
    {
        static void Main()
        {
```

```
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                ioTCore.CreateDataElement(struct1,
                    "float1",
                    GetFloat1,
                    SetFloat1,
                    true,
                    true,
                    true,
                    format: new FloatFormat(new FloatValuation(-9.9f, 9.9f, 3)));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static float GetFloat1(IBaseElement sender)
        {
            return _float1;
        }

        private static bool SetFloat1(IBaseElement sender, float value)
        {
            if (_float1.EqualsWithPrecision(value)) return false;
            _float1 = value;
            return true;
        }
        private static float _float1 = 1.2345f;
    }
}
```

1.3.4.4 Add a string data element

To add a new string data element to the IoTCore, create an object of the class “DataElement” with a string format specifier and add it to the IoTCore element tree.

```
namespace Sample08
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.Valuations;
    using ifmIoTCore.Utilities;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                ioTCore.CreateDataElement(struct1,
                    "string1",
                    GetString1,
                    SetString1,
```

```
                    true,
                    true,
                    true,
                    format: new StringFormat(new StringValuation(0, 100)));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static string GetString1(IBaseElement sender)
        {
            return _string1;
        }

        private static bool SetString1(IBaseElement sender, string value)
        {
            if (_string1 == value) return false;
            _string1 = value;
            return true;
        }
        private static string _string1 = "Hallo";
    }
}
```

1.3.4.5 Add an integer-enumeration data element

To add a new integer-enumeration data element to the IoTCore, create an object of the class “DataElement” with an integer-enumeration format specifier and add it to the IoTCore element tree.

```
namespace Sample09
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.Valuations;
    using ifmIoTCore.Utilities;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                ioTCore.CreateDataElement(struct1,
                    "enum1",
                    GetEnum1,
                    SetEnum1,
                    true,
                    true,
                    true,
                    format: new IntegerEnumFormat(new IntegerEnumValuation(
                        new Dictionary<string, string>
                        {
                            {"0", "null"},
                            {"1", "one"},
                            {"2", "two"},
                            {"3", "three"}
                        }, _enum1)));
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static int GetEnum1(IBaseElement sender)
        {
            return _enum1;
        }

        private static bool SetEnum1(IBaseElement sender, int value)
        {
            if (_enum1 == value) return false;
            _enum1 = value;
            return true;
        }
        private static int _enum1 = 1;
    }
}
```

1.3.4.6 Add an array data element

To add a new array data element to the IoTCore, create an object of the class “DataElement” with an array type format specifier and add it to the IoTCore element tree.

```
namespace Sample11
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.Valuations;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                ioTCore.CreateDataElement(struct1,
                    "array1",
                    GetArray1,
                    SetArray1,
                    true,
                    true,
                    true,
                    format: new ArrayFormat(new ArrayValuation(Format.Types.Number,
                        new IntegerFormat(new IntegerValuation(0, 100)))));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static int[] GetArray1(IBaseElement sender)
        {
            return _array1;
        }

        private static bool SetArray1(IBaseElement sender, int[] value)
```

```
        {
            if (_array1.SequenceEqual(value)) return false;
            _array1 = value;
            return true;
        }
        private static int[] _array1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    }
}
```

1.3.4.7 Add an object data element

To add a new object data element to the IoTCore, you first create the field objects, which are the members of the object data element. Then create an object of the class “DataElement” with an object format specifier, pass in the previously create fields and add it to the IoTCore element tree.

```
namespace Sample12
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.Valuations;

    internal class Program
    {
        internal class UserData : IEquatable<UserData>
        {
            public string String1;
            public int Int1;
            public float Float1;

            public UserData()
            {
                String1 = "Hallo";
                Int1 = 10;
                Float1 = 1.2345f;
            }

            public bool Equals(UserData other)
            {
                if (other == null) return false;
                return Int1 == other.Int1 &&
                Float1.Equals(other.Float1) &&
                String1 == other.String1;
            }
        }

        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                  new List<string> { "profile1" });

                var intField = new Field("intField1",
                  new IntegerFormat(new IntegerValuation(-100, 100)));
                var floatField = new Field("floatField1",
                  new FloatFormat(new FloatValuation(-100.0f, 100.0f, 3)));
                var stringField = new Field("stringField1",
                  new StringFormat(new StringValuation(10, 10, "dd-mm-yyyy")));
                var object1 = ioTCore.CreateDataElement(struct1,
                    "object1",
                    GetObject1,
                    SetObject1,
                    true,
                    true,
                    true,
```

```
                format: new ObjectFormat(new ObjectValuation(new List<Field>
                {
                    intField,
                    floatField,
                    stringField
                })));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static UserData GetObject1(IBaseElement sender)
        {
            return _object1;
        }

        private static bool SetObject1(IBaseElement sender, UserData value)
        {
            if (_object1.Equals(value)) return false;
            _object1 = value;
            return true;
        }
        private static UserData _object1 = new UserData();
    }
}
```

1.4 GETTING ELEMENTS BY ADDRESS

To get a specific element from the IoTCore element tree you can use the element address. The address of an element is the address of its parent joined with its own identifier, separated by a slash '/'. To guarantee unique addresses for the elements the element identifier must be unique inside one hierarchy level of the tree.

```
namespace Sample13
{
    using System;
    using ifmIoTCore;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1");
                ioTCore.CreateDataElement<string>(struct1, "string1");

                var string1 = ioTCore.GetElementByAddress("/struct1/string1");
                Console.WriteLine(string1.Address);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
}
```

1.5 ADD USER SPECIFIC DATA TO ELEMENTS

Each element has a member field of type “object” called “UserData”. This field allows you to link any user specific object to the element. When the service element calls the service handler, it passes a reference to itself to the service handler as first parameter. Through this reference, the service handler function can access to the user data field of the element. This can be useful, e.g. in a service handler function, when additional, element specific information is needed.

```
namespace Sample14
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Utilities;

    internal class Program
    {
        internal class UserData
        {
            public string String1;
            public int Int1;
            public float Float1;

            public UserData()
            {
                String1 = "Hallo";
                Int1 = 10;
                Float1 = 1.2345f;
            }
        }

        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    null,
                    new List<string> { "profile1" });

                var service1 = ioTCore.CreateActionServiceElement(struct1,
                    "service1",
                    HandleService1);
                service1.UserData = new UserData();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();
        }

        private static void HandleService1(IBaseElement sender, int? cid)
        {
            var userData = (UserData)sender.UserData;

            // Do something with user data
        }
    }
}
```

1.6 SUBSCRIBE TO EVENTS

1.6.1 Add an event handler to the “treechanged” event of the IoTCore

To allow your application to react when the IoTCore element tree changes, you can add an event handler to the “TreeChangedEvent” of the IoTCore root element.

```
namespace Sample15
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.Elements.EventArguments;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                // Register treechanged event handler
                ioTCore.TreeChanged += HandleTreeChangedEvent;

                // Create an element and do not raise a treechanged event
                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1",
                    raiseTreeChanged: false);

                // Raise a treechanged event on demand
                ioTCore.RaiseTreeChanged(ioTCore.Root,
                    struct1,
                    TreeChangedAction.ElementAdded);

                // Remove an element and raise a treechanged event
                ioTCore.RemoveElement(ioTCore.Root,
                    struct1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static void HandleTreeChangedEvent(object sender, EventArgs e)
        {
            var treeChangedEventArgs = (TreeChangedEventArgs)e;

            // Handle event
            switch (treeChangedEventArgs.Action)
            {
                case TreeChangedAction.ElementAdded:
                    Console.WriteLine("Element added");
                    break;
                case TreeChangedAction.ElementRemoved:
                    Console.WriteLine("Element removed");
                    break;
                case TreeChangedAction.TreeChanged:
                    Console.WriteLine("Tree changed");
                    break;
            }
        }
    }
}
```

1.6.2 Add an event handler to a “notification” event of an event element

To allow your application to react when an event is raised, you can add an event handler to the event element.

```
namespace Sample16
{
    using System;
    using System.Threading;
    using ifmIoTCore;
    using ifmIoTCore.Elements;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1");

                var event1 = ioTCore.CreateEventElement(struct1, "event1");

                event1.EventHandler += HandleNotificationEvent;

                var eventTimer = new Timer(TriggerEvent1, event1, 5000, 60000);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static void HandleNotificationEvent(object sender, EventArgs e)
        {
            Console.WriteLine("HandleNotificationEvent called");
        }

        private static void TriggerEvent1(object param)
        {
            Console.WriteLine("TriggerEvent1 called");

            var event1 = (IEventElement)param;
            event1.RaiseEvent();
        }
    }
}
```

1.6.3 Add a subscription to an event element

To request the values of multiple data elements together with an event you can add a subscription to an event element and add an event handler to the event element. If you set the “callback” field in the subscription to an URI, then IoTCore will send the event to the remote service specified in the URI, if you set the “callback” field to the address of a local service element, then the service handler for that service will be called.

```
namespace Sample17
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.ServiceData.Events;
    using ifmIoTCore.Elements.ServiceData.Requests;
    using ifmIoTCore.Elements.Valuations;
    using ifmIoTCore.Utilities;

    internal class Program
    {
        static void Main()
        {
            try
```

```
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1");

                ioTCore.CreateSetterServiceElement<EventServiceData>(struct1,
                    "service1",
                    HandleService1);

                var string1 = ioTCore.CreateDataElement(struct1,
                    "string1",
                    GetString1,
                    null,
                    true,
                    false,
                    format: new StringFormat(new StringValuation(0, 100)));

                var event1 = ioTCore.CreateEventElement(struct1,
                    "event1");

                event1.Subscribe(new SubscribeRequestServiceData("/struct1/service1",
                  new List<string> { string1.Address }));

                event1.RaiseEvent();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static string GetString1(IBaseElement sender)
        {
            return _string1;
        }
        private static string _string1 = "Hallo";

        private static void HandleService1(IBaseElement sender,
            EventServiceData data, int? cid = null)
        {
            Console.WriteLine("HandleService1 called");

            Console.WriteLine($"Event source={data.EventSource}");
            Console.WriteLine($"Event number={data.EventNumber}");
            foreach (var (key, value) in data.Payload)
            {
                Console.WriteLine($"{key}={value}");
            }
        }
    }
}
```

1.7 RAISE EVENTS

You can raise an event to inform your application about an incident in your application. To raise an event called the “RaiseEvent” method of your event element.

```
namespace Sample18
{
    using System;
    using System.Threading;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Utilities;

    internal class Program
    {
        static void Main()
        {
```

```
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = ioTCore.CreateStructureElement(ioTCore.Root,
                    "struct1");

                var event1 = ioTCore.CreateEventElement(struct1,
                    "event1");

                event1.EventHandler += HandleNotificationEvent;

                var eventTimer = new Timer(TriggerEvent1, event1, 5000, 60000);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static void HandleNotificationEvent(object sender, EventArgs e)
        {
            Console.WriteLine("HandleNotificationEvent called");
        }

        private static void TriggerEvent1(object param)
        {
            Console.WriteLine("TriggerEvent1 called");

            var event1 = (IEventElement)param;
            event1.RaiseEvent();
        }
    }
}
```

1.8 CREATE NETWORK ADAPTERS

1.8.1 Create a server network adapter

To make your IoTCore accessible from another program over a network you must create a network adapter server and register it with your IoTCore, and then start the server. Network adapter servers can support various protocols over various communication channels. The standard, what all IoTCores support is JSON over HTTP.

```
namespace Sample19
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.NetAdapter.Http;

    internal class Program
    {
        static void Main()
        {
            HttpServerNetAdapter httpServer = null;
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                httpServer = new HttpServerNetAdapter(ioTCore,
                    new Uri("http://127.0.0.1:8001"),
                    new ifmIoTCore.Converter.Json.JsonConverter());

                ioTCore.RegisterServerNetAdapter(httpServer);

                httpServer.Start();
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
            httpServer?.Stop();
        }
    }
}
```

1.8.2 Create a client network adapter factory

To allow your IoTCore to connect to other IoTCores over a network you must create a network adapter client factory and register it with your IoTCore. This is necessary for instance to enable your IoTCore to send events to another IoTCore or to mirror another IoTCore into its own element tree. When your IoTCore needs to connect to another IoTCore it uses the network adapter client factory to create a network adapter client. With this client it connects to the remote IoTCore.

```
namespace Sample20
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.NetAdapter.Http;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                ioTCore.RegisterClientNetAdapterFactory(
                    new HttpClientNetAdapterFactory(
                        new ifmIoTCore.Converter.Json.JsonConverter()));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();
        }
    }
}
```

1.8.3 Create a mqtt server network adapter

Like in the http example there is the possibility to create a mqtt network adapter server. In the constructor the topic parameter will be the topic on which the server will be subscribed on the broker to listen for commands.

```
namespace DemoMqtt
{
        using System;
        using System.Net;
        using System.Threading;
        using ifmIoTCore;
        using ifmIoTCore.Converter.Json;
        using ifmIoTCore.Elements;
        using ifmIoTCore.NetAdapter.Http;
        using ifmIoTCore.NetAdapter.Mqtt;

        class Program
        {
            static void Main(string[] args)
            {
                var iotCore = IoTCoreFactory.Create("test");
```

```
                MqttServerNetAdapter mqttServerNetAdapter = new MqttServerNetAdapter(iotCore, new
JsonConverter(), new IPEndPoint(IPAddress.Loopback, 1883), DisconnectionHandler, "commandTopic");
                iotCore.RegisterServerNetAdapter(mqttServerNetAdapter);

                HttpServerNetAdapter httpServerNetAdapter = new HttpServerNetAdapter(iotCore, new
Uri("http://127.0.0.1:8090"),new JsonConverter());
                iotCore.RegisterServerNetAdapter(httpServerNetAdapter);

                httpServerNetAdapter.Start();
                mqttServerNetAdapter.Start();

                mqttServerNetAdapter.EventMessageReceived += (sender, e) =>
                {
                    iotCore.HandleEvent(e.EventMessage);
                };

                mqttServerNetAdapter.RequestMessageReceived += (sender, e) =>
                {
                    e.Response = iotCore.HandleRequest(e.Request);
                };

                using (var manualResetEvent = new ManualResetEventSlim()) {

                    var stopServiceElement = new ActionServiceElement(((element, i) =>
                    {
                        manualResetEvent.Set();
                    }), "stop");

                    iotCore.Root.AddChild(stopServiceElement);

                    manualResetEvent.Wait();
                }

                httpServerNetAdapter.Dispose();
                mqttServerNetAdapter.Dispose();
            }

            private static void DisconnectionHandler(Exception exception)
            {
                // Handle disconnection here, for instance by calling mqttServerNetAdapter.Start()
again.
            }
        }
 }
```

1.8.3.1 Server netadapter iotcore structure

When a server netadapter is created following structure will be created in the iotcore tree:

An item which implements the comminterface profile will be added for this connection under the /connections Element in the root element. The comminterface will have a name like:

mqttconnection_F2FD1939-DF78-4777-855E-8AE11C2A1008

where the guid part is unique for every connection.

This element will contain:

1) a mqttsetup element. With this element the qos and the version of the mqtt connection can be read. Qos can also be set.
2) a status element of type runcontrol with which the netadapter server can be started and stopped. The status element itself is of type dataelement and returns the current state of the netadapter, which can be one of “stopped”, “running”, “init”, “error”.
3) a mqttcmdchannel element of type commchannel. The mqttcmdchannel element will contain a mqttcmdchannelsetup element. With this element the user can set the brokerip, brokerport, commandTopic and replyTopic of the mqtt server netadapter connection.

The final structure is then like this (indentation reflects containment):

```
"identifier": "connections",
            "identifier": " mqttconnection_F2FD1939-DF78-4777-855E-8AE11C2A1008",
            "type": "structure",
            "profiles": [
              "commInterface"
            ],
                "identifier": "type",
                                "type": "data",
                    "identifier": "getdata",
                                           "type": "service"
                "identifier": "status",
                "type": "data",
                "profiles": [
                  "runcontrol"
                ]
                    "identifier": "getdata",
                    "type": "service"

                    "identifier": "start",
                    "type": "service"

                    "identifier": "stop",
                    "type": "service"

                    "identifier": "reset",
                    "type": "service"

                    "identifier": "preset",
                    "type": "data",

                        "identifier": "getdata",
                        "type": "service"

                        "identifier": "datachanged",
                        "type": "event",

                            "identifier": "subscribe",
                            "type": "service"

                            "identifier": "unsubscribe",
                            "type": "service"

                            "identifier": "getsubscriptioninfo",
                            "type": "service"

                    "identifier": "datachanged",
                    "type": "event",

                        "identifier": "subscribe",
                        "type": "service"

                        "identifier": "unsubscribe",
                        "type": "service"

                        "identifier": "getsubscriptioninfo",
                        "type": "service"

                "identifier": "MQTTSetup",
                "type": "structure",
                "profiles": [
                  "mqttSetup"
                ],
                    "identifier": "QoS",
                    "type": "data",
                    "profiles": [
                      "parameter"
                    ],

                        "identifier": "setdata",
                        "type": "service"

                        "identifier": "getdata",
                        "type": "service"
```

```
                    "identifier": "datachanged",
                    "type": "event",

                        "identifier": "subscribe",
                        "type": "service"

                        "identifier": "unsubscribe",
                        "type": "service"

                        "identifier": "getsubscriptioninfo",
                        "type": "service"

                "identifier": "version",
                "type": "data",
                "profiles": [
                  "parameter"
                ],

                    "identifier": "getdata",
                    "type": "service"

            "identifier": "mqttCmdChannel",
            "type": "structure",
            "profiles": [
              "commChannel"
            ],

                "identifier": "type",
                "type": "data",
                "profiles": [
                  "parameter"
                ],

                    "identifier": "getdata",
                    "type": "service"

                "identifier": "status",
                "type": "data",
                "profiles": [
                  "runcontrol"
                ],

                    "identifier": "getdata",
                    "type": "service"

                    "identifier": "start",
                    "type": "service"

                    "identifier": "stop",
                    "type": "service"

                    "identifier": "reset",
                    "type": "service"

                    "identifier": "preset",
                    "type": "data",

                        "identifier": "getdata",
                        "type": "service"

                        "identifier": "datachanged",
                        "type": "event",

                            "identifier": "subscribe",
                            "type": "service"

                            "identifier": "unsubscribe",
                            "type": "service"

                            "identifier": "getsubscriptioninfo",
                            "type": "service"

                    "identifier": "datachanged",
                    "type": "event",
```

```
                "identifier": "subscribe",
                "type": "service"

                "identifier": "unsubscribe",
                "type": "service"

                "identifier": "getsubscriptioninfo",
                "type": "service"

        "identifier": "mqttCmdChannelSetup",
        "type": "structure",
        "profiles": [
          "mqttCmdChannelSetup"
        ],

            "identifier": "brokerIP",
            "type": "data",
            "profiles": [
              "parameter"
            ],

                "identifier": "setdata",
                "type": "service"

                "identifier": "getdata",
                "type": "service"

                "identifier": "datachanged",
                "type": "event",

                    "identifier": "subscribe",
                    "type": "service"

                    "identifier": "unsubscribe",
                    "type": "service"

                    "identifier": "getsubscriptioninfo",
                    "type": "service"

            "identifier": "cmdTopic",
            "type": "data",
            "profiles": [
              "parameter"
            ],

                "identifier": "setdata",
                "type": "service"

                "identifier": "getdata",
                "type": "service"

                "identifier": "datachanged",
                "type": "event",

                    "identifier": "subscribe",
                    "type": "service"

                    "identifier": "unsubscribe",
                    "type": "service"

                    "identifier": "getsubscriptioninfo",
                    "type": "service"

            "identifier": "brokerPort",
            "type": "data",
            "profiles": [
              "parameter"
            ],
                "identifier": "setdata",
                "type": "service"

                "identifier": "getdata",
                "type": "service"

                "identifier": "datachanged",
                "type": "event",
```

```
                                "identifier": "subscribe",
                                "type": "service"

                                "identifier": "unsubscribe",
                                "type": "service"

                                "identifier": "getsubscriptioninfo",
                                "type": "service"

                        "identifier": "defaultReplyTopic",
                        "type": "data",
                        "profiles": [
                          "parameter"
                        ],

                            "identifier": "setdata",
                            "type": "service"

                            "identifier": "getdata",
                            "type": "service"

                            "identifier": "datachanged",
                            "type": "event",
                            "subs": [
                              {
                                "identifier": "subscribe",
                                "type": "service"
                              },
                              {
                                "identifier": "unsubscribe",
                                "type": "service"
                              },
                              {
                                "identifier": "getsubscriptioninfo",
                                "type": "service"
                              }
                            ]
                          }
                        ]
                      }
                    ]
                  }
                ]
              }
            ]
          }
        ]
      }
```

1.8.4 Create a mqtt client network adapter factory

To enable the IoTCore to dynamically create mqtt client adapters, a mqtt client factory is registered.

```
namespace DemoMqtt
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.Converter.Json;
    using ifmIoTCore.NetAdapter.Mqtt;

    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                var iotCore = IoTCoreFactory.Create("test");
                iotCore.RegisterClientNetAdapterFactory(new MqttNetAdapterClientFactory(new JsonCon-
verter()));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
```

```
            }

            Console.ReadLine();
        }
    }
}
```

1.8.4.1 Mqtt callback address

When subscribing to events in the iotcore a mqtt schemed callback address can be given.

Therefor a `MqttNetAdapterClientFactory` must be registered like in the previous example.

Then an address like
mqtt://192.168.1.1/target/topic

can be used to direct the event to the mqtt broker at 192.168.1.1 and the event will be published with the topic: /target/topic.

1.8.4.2 Sending Requests and Events

The user can make use of the SendRequest method, to send request, and receive responses from another iotcore instance.

An example for the mqtt client netadapter would look like this:

```
IClientNetAdapter client = new MqttClientNetAdapter(
                "cmdTopic",
                new IPEndPoint(
                    IPAddress.Parse("192.168.1.1"),
                    1883),
                new JsonConverter(),
                TimeSpan.FromSeconds(1));

client.SendRequest(new RequestMessage(10, "/gettree", new GetTreeRequestServiceData("/",
0).ToJson()));
```

Accordingly with the SendEvent method the user can send events:

```
client.SendEvent(
                new EventMessage(10, "/serviceHandler",
                    new EventServiceData("/identifier", 0,
                        new Dictionary<string, JToken>
                        {
                            {
                                "/identifier",
                                JToken.FromObject(3)
                            }
                        }).ToJson()));
```

1.8.5 Create an OPC UA server network adapter

To make your IoTCore accessible via OPC UA you can create an OPC UA server network adapter and register it with your IoTCore. The OPC UA server network adapter maps all elements from your IoTCore into the address space of the OPC UA server.

The OPC UA server uses the following IoTCore element type to OPC UA node type mapping:

- A device element is mapped to an object node
- A structure element is mapped to a folder node
- A data element is mapped to a variable node

- A service element is mapped to a method node
- An event element is mapped to an object node

If the IoTCore raises a tree-changed event, the OPC UA server synchronizes its nodes with the IoTCore element tree.

```
namespace Sample21
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.NetAdapter.Opc;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var opcServer = OpcServerNetAdapter.CreateInstance(ioTCore,
                    ioTCore.Root,
                    ioTCore.DataStore,
                    ioTCore.Logger);

                opcServer.Uri = new Uri("http://127.0.0.1:62546");

                ioTCore.RegisterServerNetAdapter(opcServer);

                opcServer.Start();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
}
```

1.8.6 Create a CANopen server network adapter

Like in the http example there is the possibility to create a CANopen network adapter server. In the constructor you must set the same parameter like in the http example. You need an IoTCore object a CAN - Uri (**1.8.8.1**) and the.

```
using System;
using ifm.CiA301.Can.Adapter;
using ifmIoTCore;
using ifmIoTCore.Messages;
using ifmIoTCore.NetAdapter.CanOpen;
using ifmIoTCore.Converter.Json;
using ifmIoTCore.NetAdapter.CanOpen.UriBuilders;

namespace Sample22
{
    using System;

    internal class Program
    {
        static void Main()
        {
            try
            {
                // Create a base iotcore instance.
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var uri = new EcomatControllerCanOpenUriBuilder(Vendor.Sontheim.ToString(),
"canfox(1) ch01", 125, 127, 1200, "Optimal");
```

```
                // Create a CANopen server NetAdapter a client can access to.
                var serverNetAdapter = new CanOpenServerNetAdapter(ioTCore, uri, new JsonCon-
verter());

                // Register the NetAdapterServer on the current IoTCore and run the server.
                ioTCore.RegisterServerNetAdapter(serverNetAdapter);
                var task = serverNetAdapter.StartAsync();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();
        }
    }
}
```

1.8.7 Create a CANopen client network adapter

To have access to a CANopen server NetAdapter, we need to create the client part.

```
using System;
using ifm.CiA301.Can.Adapter;
using ifmIoTCore;
using ifmIoTCore.Messages;
using ifmIoTCore.NetAdapter.CanOpen;
using ifmIoTCore.Converter.Json;
using ifmIoTCore.NetAdapter.CanOpen.UriBuilders;

namespace Sample22
{
    using System;

    internal class Program
    {
        static void Main()
        {
            try
            {
                // Create a base iotcore instance.
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var uri = new EcomatControllerCanOpenUriBuilder(Vendor.Sontheim.ToString(),
"canfox(2) ch01", 125, 127, 1200, "Optimal");

                // Create the CANopen client NetAdapter to have direct access to a NetAdapterServer.
                var clientNetAdapter = new CanOpenClientNetAdapter(uri, new JsonConverter());

                // Create a /gettree request and send the request.
                var request = new RequestMessage(0, "/gettree", null);
                var response = clientNetAdapter.SendRequest(request, new TimeSpan(0,0,5));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();
        }
    }
}
```

1.8.8 Create a CANopen client network adapter factory

Like described in the http example. To enable the IoTCore to dynamically create CANopen client adapters, a CANopen client factory must be registered. Later on you can do mirroring and the CANopen client will be create dynamically by the IoTCore.

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

```
using ifm.CiA301.Can.Adapter;
using ifmIoTCore;
using ifmIoTCore.Messages;
using ifmIoTCore.NetAdapter.CanOpen;
using ifmIoTCore.Converter.Json;

namespace Sample22
{
    using System;

    internal class Program
    {
        // This is a CanOpen simple sample.
        static void Main()
        {
            try
            {
                // Create a base iotcore instance.
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");


                ioTCore.RegisterClientNetAdapterFactory(new CanOpenNetAdapterClientFactory(new
JsonConverter()));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();
        }
    }
}
```

1.8.8.1 Configure a CANopen server/client network adapter/factory

The Uri and the CAN-Adapter parameter are different from the http example.

a) The Uri definition of a CANopen network adapter server/client or factory must implement the following values:

- **scheme:** The required scheme must set to "canopen". For example: canopen:///
- **host:** The required host must set to an empty string. For example: canopen:///
- **some parameter :**
  - **nodeid:** The required nodeid parameter. For example: canopen:///?nodeid=116
  - **index:** The required index parameter. For example: canopen:///?index=8016
  - **subindexdownload:** The required subindexdownload parameter. For example: canopen:///?subindexdownload=1
  - **subindexupload:** The required subindexupload parameter. For example: canopen:///?subindexupload=2
  - **timeout:** The optional timeout parameter. For example: canopen:///?timeout=5000
  - **compression:** The optional compression parameter (Optimal, Fastest, NoCompression). For example: canopen:///?compression=Optimal

  Note: The Uri for a server/client network adapter must be the same. Only the adaptename must be different. If not, a connection between this server and client is not possible! If a server is implemented and a client is registered via the client factory, it is possible for the IoTCore to create CANopen client network adapter for example via mirroring. In this case, the Uri parameter of the mirror request must contain the same Uri like the server network adapter exclude the adaptername!

b) The CAN – Adapter object:

Supported CAN -Adapter Vendors are IXXAT, PEAK, Kvaser and Sontheim for Windows and ifm socket CAN for ifm ecomat display devices. Please make sure that you have installed the drivers if necessary. To get available CAN – Adapter create the specific CAN - Adapter object (namespace: ifm.CiA301.Can.Adapter) and call the GetAdapters() function to get a list of available CAN – Adapter.

1.9 LOGGING

The IoTCore supports logging. You can enable or disable logging when you create an IoTCore with the "IoTCoreFactory" class. In the IoTCore factory are several overloaded methods available that allow to create different types of loggers. You can enable or disable the default logger and set a log level, you can also provide your own logger class by passing a logger object that implements the "ILog" interface into the create method of the IoTCore factory or you can pass a logger configuration file to configure the default logger. The log file for the default logger is called "IoTCore.log" and is created in the "%ProgramData%\ifm\IoTCore\logs" directory.

```
namespace Sample23
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.Logging.Log4Net;
    using ifmIoTCore.Utilities;

    internal class Program
    {
        static void Main()
        {
            try
            {
                var logger = new Logger(LogLevel.Info);
                var ioTCore = IoTCoreFactory.Create("MyIoTCore",
                    logger);

                logger.Info("Informational log");

                ioTCore.Logger.Info("Call logger via IoTCore");
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
}
```

1.10 ERROR HANDLING AND ERROR CODES

For error conditions, that the IoTCore handles it throws an "IoTCoreException". The "IoTCoreException" is derived from the standard .NET exception class. It reports an error code, an error message and optional error details. For error conditions, which are detected by the underlying .NET runtime it throws a .NET exception.

The following error codes are reported by IoTCore:

Code	Meaning	Description
400	Bad request	The request is invalid or malformed
404	Not found	The requested element does not exist
423	Locked	The service is temporarily locked, disabled or busy
500	Internal error	An error condition, that is not considered and handled by IoTCore and therefore raised by the underlying runtime
501	Not implemented	The service is not implemented

530	Data invalid	The provided data is invalid
901	Already exists	The element already exists
902	Not allowed	The operation is not allowed

1.11 Create and Load an IoTCore Profile

The IoTCore provides an extension interface through which the IoTCore functionality can be extended with loadable plugins, which implement so called profiles. The interface is called “IProfileBuilder” and is defined like this:

```
public interface IProfileBuilder : IDisposable
{
    /// <summary>
    /// Gets the name of the profile.
    /// </summary>
    string Name { get; }

    /// <summary>
    /// Builds the profile into the tree.
    /// </summary>
    void Build();
}
```

An implementation of the interface can use the API of the IoTCore to add its own elements to the element tree and connect its own services to the elements. Here is an example:

```
namespace Sample27
{
    using System;
    using System.Collections.Generic;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.NetAdapter.Http;
    using ifmIoTCore.Utilities;
    using Newtonsoft.Json;

    public class MyRequestServiceData
    {
        [JsonProperty("in", Required = Required.Always)]
        public readonly string Input;

        public MyRequestServiceData(string input)
        {
            Input = input;
        }
    }

    public class MyResponseServiceData
    {
        [JsonProperty("out", Required = Required.Always)]
        public readonly string Output;

        [JsonProperty("length", Required = Required.Always)]
        public readonly int Length;

        public MyResponseServiceData(string output, int length)
        {
            Output = output;
            Length = length;
        }
    }

    public class MyProfile
    {
        private readonly IIoTCore _ioTCore;

        public string Name = "MyProfile";
```

```
        public MyProfile(IIoTCore ioTCore)
        {
            _ioTCore = ioTCore;
        }

        public void Build()
        {
            var structureElement = _ioTCore.CreateStructureElement(_ioTCore.Root,
                "my_profile",
                profiles: new List<string> {"my_profile"} );
            _ioTCore.CreateServiceElement<MyRequestServiceData, MyResponseServiceData>(structureEl-
ement,
                "do_it",
                DoItFunc,
                raiseTreeChanged: false);
        }

        private MyResponseServiceData DoItFunc(IBaseElement element,
            MyRequestServiceData data,
            int? cid = null)
        {
            return DoIt(data);
        }

        // Make service available on API
        public MyResponseServiceData DoIt(MyRequestServiceData data)
        {
            return new MyResponseServiceData(data.Input, data.Input.Length);
        }

        public void Dispose()
        {
        }
    }

    class Program
    {
        static void Main()
        {
            var ioTCore = IoTCoreFactory.Create("MyIoTCore");

            var myProfile = new MyProfile(ioTCore);
            myProfile.Build();

            // Call service via API
            var response = myProfile.DoIt(new MyRequestServiceData("The little red rooster"));
            Console.WriteLine(Helpers.ToJson(response));

            // Call service via element
            var element = (IServiceElement)ioTCore.Root.GetElementByIdentifier("do_it");
            var json = element.Invoke(Helpers.ToJson(new MyRequestServiceData("The little red
rooster")),
                null);
            Console.WriteLine(json);

            element = (IServiceElement)ioTCore.Root.GetElementByAddress("/my_profile/do_it");
            json = element.Invoke(Helpers.ToJson(new MyRequestServiceData("The little red
rooster")),
                null);
            Console.WriteLine(json);

            // Make service accessible from remote under /my_profile/do_it
            var httpServer = new HttpServerNetAdapter(ioTCore,
                new Uri("http://127.0.0.1:8101"),
                new ifmIoTCore.Converter.Json.JsonConverter());

            httpServer.Start();

            Console.ReadLine();
        }
    }
}
```

As this sample shows the service method can be called from the application either directly or via the service element in the IoTCore tree. By starting an network adapter server the service is also accessible from remote, in this case via http.

The required JSON request would be

```
{
    "code":10,
    "cid": 10,
    "adr": "/my_profile/do_it",
    "data":
    {
        in: "The little red rooster"
    }
}
```

The response would be

```
{
    "code": 200,
    "cid": 10,
    "adr": "/my_profile/do_it",
    "data":
    {
        "out": "Received: The little red rooster",
        "length": 22
    }
}
```

2 CLASS REFERENCE

You find the complete class reference in the archive “class_reference.zip”, which you find in the IoTCore.Net package. To view the class reference in your browser, open the file “class_reference\html\index.html”.