ifm electronic

IoTCore.Net Programming Guide

Version 2.0

Revision	Comment	Date	Author
1.0	Initial creation	2020	Johannes Schuster Aleksander Mihajlovic Nick Vetter
2.0	Updated to version 2.0	2022/4/29	Johannes Schuster

CONTENTS

1 WHAT IS THE IOTCORE.NET?

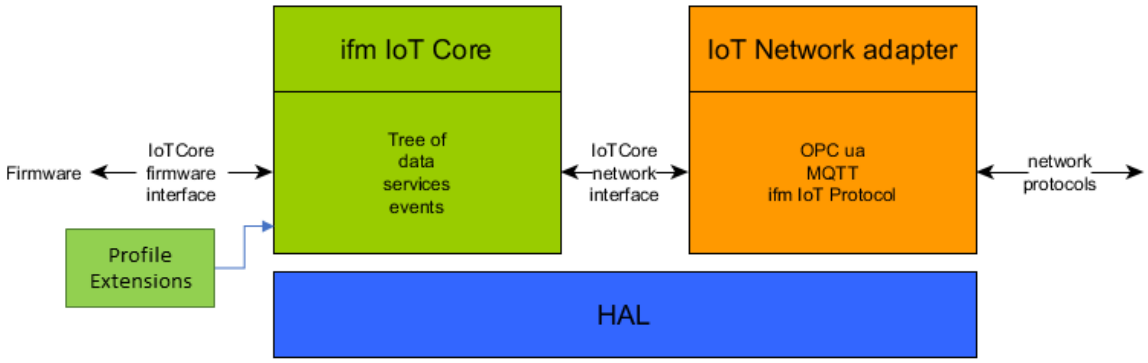
The IoTCore.Net is an application framework, that allows an application to organize its data, services, and events in a hierarchical structure of elements. An element is the basic object to represent the data, services, and events. These elements can be exposed to external applications through network adapters. The IoTCore.Net implements the ifm IoTCore communication specification to enable devices running an IoTCore.Net application to communicate with other applications and devices.

1.1 TARGET PLATFORM

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

2 CONCEPTS

2.1 THE IOTCORE ARCHITECTURE



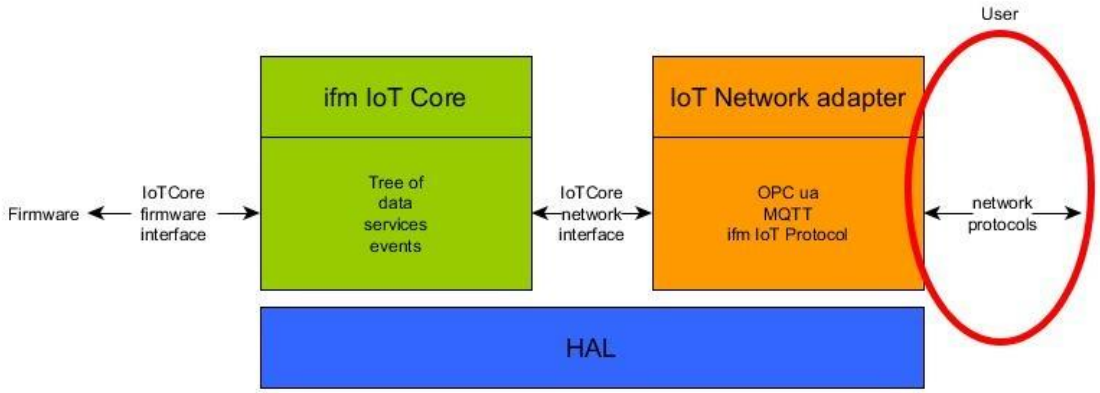
The IoTCore itself implements just a basic set of standard services, but it provides several interfaces through which an application can extend the functionality of the IoTCore. The IoTCore also provides interfaces through which an application can load network adapter servers into the IoTCore to make the application accessible from external applications. To make the picture complete the IoTCore also offers interfaces through with network adapter clients can be loaded into the IoTCore to enable the IoTCore to communicate with other external applications.

The standard set of services provided by the IoTCore can be extended by the application by implementing its own data, service, and event elements and it can also load already implemented and ready to use modules that implement well known and widely used profiles. Profiles are, like a contract, a predefined set of data, service, and event elements.

The IoTCore network adapter interfaces provide interfaces to load network adapter plugins, which can be loaded by the application into the IoTCore. Each network adapter implements its own communication channel and protocol. The network adapter servers provide the application's services to external applications, whereas the network adapter clients enable the IoTCore to communicate with external applications, like other IoTCores or MQTT message brokers, for instance.

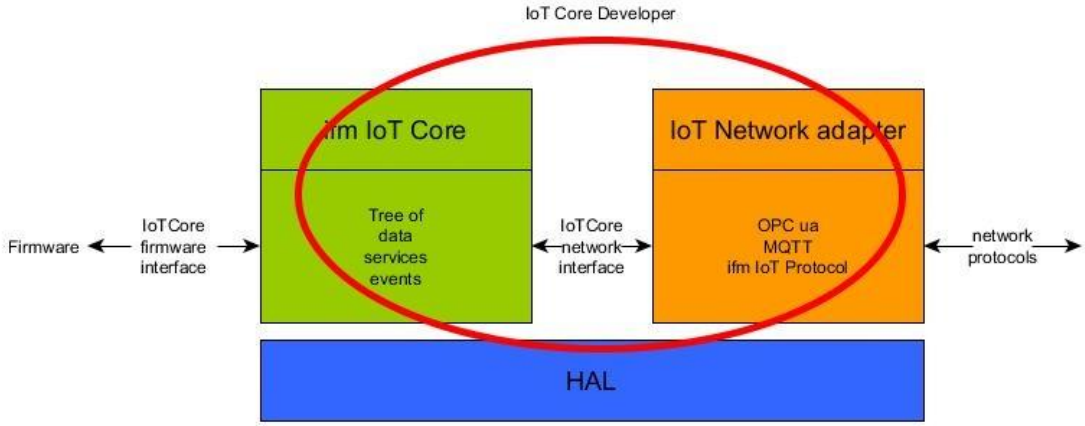
2.2 USING THE IOTCORE

Users of an IoTCore application communicate with the IoTCore via the network protocol API. Users can be, for instance, other applications that control the IoTCore application or websites that allow monitoring the status of the IoTCore application.



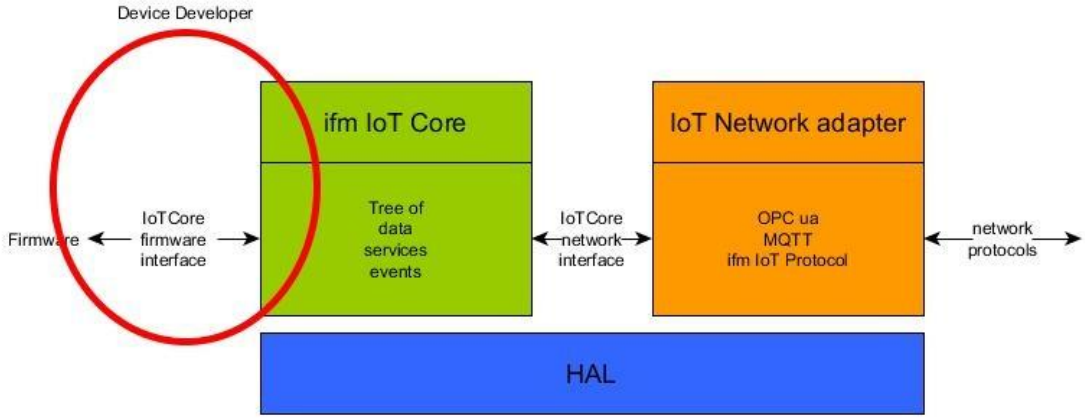
2.3 DEVELOPING THE IOTCORE

The IoTCore developers naturally work on the IoTCore, network adapters, protocols, and profiles. As the IoTCore supports an open architecture, every user can write its own set of profiles or network adapter for various communication hardware and transport layers and communication protocols.



2.4 INTEGRATING THE IOTCORE INTO AN APPLICATION

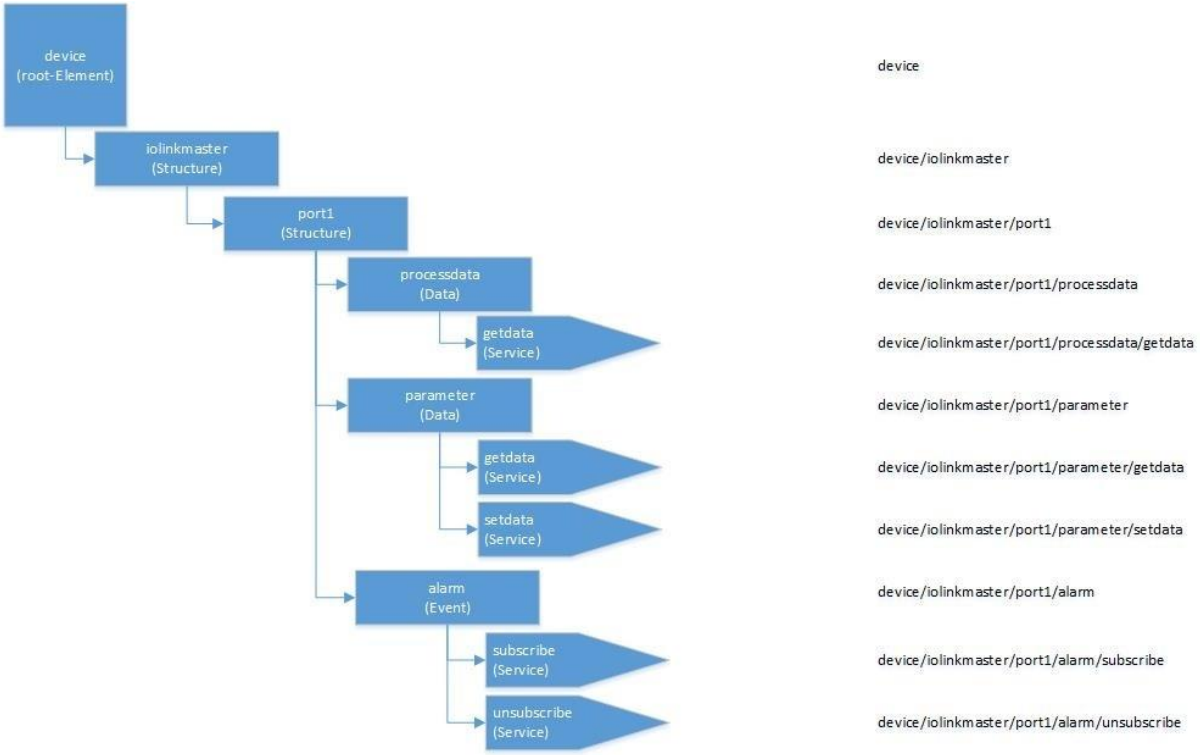
Integrators of an IoTCore application typically implement the data, service, and events for an application and create the corresponding elements in the IoTCore.



2.5 Modelling Data, Services and Events

The purpose of the IoTCore is to represent the data and functionality of an application and make it available to other applications or users. The application is modeled as a structured tree of elements, where each element can be of type service, data, event, or structure with a device element at the root of the element tree. The root element is always a device element. The name “device” has historical reasons, but an IoTCore application can be anything and is not limited to a kind of physical device.

The application, in this view, is a tree of elements describing all the functionality of the application in form of data, services, and events (supported by structure elements to organize the tree).



2.6 Addressing Elements

Each element in the IoTCore is referenced by its 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. The same identifier is allowed in different branches of the tree. The name of the root device element is optional.

Examples:

device/iolinkmaster/port1/processdata/getdata

/iolinkmaster/port1/processdata/getdata

2.7 Interacting with the IoTCore

The IoTCore provides two interfaces, on the one side the application programming interface (API), which is used by applications or firmware to implement the device functionality and on the other side the command interface which is used by network adapters.

The target for any request to the IoTCore is the service element. The service element is referenced by its address and the request with its parameters is passed to the service element for processing. The service element returns a response with its parameters.

The IoTCore can raise events on behalf of the application. The source of an event is the event element. Consumers can subscribe to the event to get notified when the event fires.

2.8 THE COMMAND INTERFACE

The IoTCore provides a command interface through messages, that allows network adapters to communicate with the IoTCore. The command interface supports a synchronous request-response communication model. The data is exchanged in form of messages. When a client calls a **request**, the IoTCore delegates the request to the addressed service element which processes the request and returns a **response** to the IoTCore, which in turn returns that response to the client, that called the request. Only service elements can process a command, therefore only service elements can be addressed. If the addressed element does not exist or is not a service, the IoTCore returns an error. The parameters for a request and the data of a response or an event is provided in the "data" field of the message.

The IoTCore can also raise **event** notifications. The events are sent as an event message to the recipient of the event. An event message does not require a response.

Request message type

Field	Mandatory	Data Type	Meaning
code	yes	Integer or String	Request type
cid	yes	Integer	A unique context ID, provided in the request and returned in the corresponding response; best practice for values: -1 (not needed/ignore); 0 <= cid < int32_max
adr	yes	String	Address of the service element in the IoTCore tree which is called
data	no	Variant	The incoming data for the service request
reply	no	String	The address of the response. If this field is present, the response will be sent to this address. If not provided, the response is sent to the address that was requested.
auth	no	Authentication Info	Authentication information for the service request

Authentication Info type

Parameter	Mandatory	Data Type	Meaning
user	yes	String	The username

passwd	no	String	The password

Response message type

Field	Mandatory	Data Type	Meaning
code	yes	Integer or String	Response result
cid	yes	Integer	A unique context ID, provided in the request and returned in the corresponding response
adr	yes	String	Address of the service element in the IoTCore tree which is called or the reply adr
data	no	Variant	The outgoing data for the service response

Event message type

Field	Mandatory	Data Type	Meaning
code	yes	Integer or String	Request type
cid	yes	Integer	A unique context ID, provided in the request and returned in the response
adr	yes	String	The target address for the event as specified in the subscribe request
data	no	Event Info	The event data

Event Info type

Parameter	Mandatory	Data Type	Meaning
srcurl	yes	String	The address of the event element, that sends the event
eventno	yes	Integer	Number of the event
payload	no	<String:Variant>[]	List of data element address:value pairs, as requested in the subscription

subscribeid	yes	Integer	The identifier for the subscription

Request codes

The type of request is identified by the "code" field. The code may be given as a string or as a number.

Code as string	Code as number	Meaning
"request"	10	A request message
"transaction"	11	A request that starts/ends a transaction
"command"	12	
"event"	80	An event message

Response codes

Code	Msg (examples)	Description
200	"Success"	The service executed successfully
400	"Bad request: <Reason>"	The service request is invalid or malformed
401	"Access denied: <Address>"	Authentication is required and has failed or has not been provided.
403	"Forbidden: <Reason>"	The service request is forbidden
404	"Element not found: <Address>"	The requested service does not exist
413	"Request payload too large"	The payload of the request is too large
414	"Request-URL too long"	The URL of the request is too large
416	"Requested Range Not Satisfiable"	Value range requested is not in permissible limit
422	"Invalid payload. Invalid <Parameter>: <Value>"	The provided data/payload is invalid
423	"The service is busy"	The service is locked, disabled or busy now
424	"Failed dependency: <Reason>"	The service is currently not allowed. Example: "Failed dependency: first call <Service>"

429	"Too many requests"	Too many requests
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
502	"Remote service failed with code <Code>: <Msg>"	A request to the remote device failed with an error.
503	"Not available: <Reason>"	The service is currently not available
504	"Remote service timeout"	The remote device did not respond within the timeout.
507	"Insufficient storage"	The memory / storage of the device is insufficient
550	"Service execution failed"	The service execution failed
901	"Already exists"	The element already exists

2.8.1 Error Handling

An error is when a service request cannot be successfully processed. When an error occurs while processing a service request an appropriate error response message is created and returned to the caller.

When calling an IoTCore service the service request is first received by the network adapter which passes the request on to the IoTCore. The IoTCore looks up the target service element for the request and passes the request to that service element for processing. In each step an error can occur that does not allow further processing of the request. So, each level must be able to return an appropriate error response. The response code in the response message will indicate the type of error. Additional error information can be provided in the response message data field.

The Error Info type

Parameter	Mandatory	Datatype	Description
data/code	Optional	Number	device specific error code
data/msg	optional	String	Textual representation of error code
data/details	optional	String	Additional and more detailed information

3 THE IOTCORE.NET API

3.1 CREATE AN IOTCORE APPLICATION

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();
        }
    }
}
```

3.2 ADD ELEMENTS TO THE IOTCORE 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 IoTCore 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 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, create an element object and call the “AddChild()” method of the element to which you want to add the newly created element. The “AddChild()” method has additional parameter, that defines whether the “treechanged” event of the parent element is raised or not.

3.2.1 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;
    using ifmIoTCore.Elements;

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

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

3.2.2 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 and return outgoing data. The data type of the object that is passed into the service handler function or returned from it is of type “Variant”. 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 “GetterServiceElement” class defines a service handler function that has no incoming data but returns outgoing data. The “SetterServiceElement” 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 ifmIoTCore;
    using ifmIoTCore.Common.Variant;
    using ifmIoTCore.Elements;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var service1 = new ActionServiceElement("service1", HandleService1);
                struct1.AddChild(service1);
```

```
                var service2 = new GetterServiceElement("service2", HandleService2);
                struct1.AddChild(service2);

                var service3 = new SetterServiceElement("service3", HandleService3);
                struct1.AddChild(service3);

                var service4 = new ServiceElement("service4", HandleService4);
                struct1.AddChild(service4);

                var service5 = new ServiceElement("service5", HandleService5);
                struct1.AddChild(service5);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

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

        private static Variant HandleService2(IServiceElement element, int? cid)
        {
            Console.WriteLine("Do something and return result");
            return new VariantValue(0);
        }

        private static void HandleService3(IServiceElement element, Variant data, int? cid)
        {
            Console.WriteLine($"Do something with {(string)(VariantValue)data}");
        }

        private static Variant HandleService4(IServiceElement element, Variant data, int? cid)
        {
            Console.WriteLine($"Do something with {data} and return result");

            var str = (string)(VariantValue)data;
            return (VariantValue)$"Received {str}";
        }

        private static Variant HandleService5(IServiceElement element, Variant data, int? cid)
        {
            var value = Variant.ToObject<UserData>(data);
            Console.WriteLine($"Do something with {value} and return result");
            return Variant.FromObject(value);
        }
    }

    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}";
        }
```

```
    }
}
```

3.2.3 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 ifmIoTCore;
    using ifmIoTCore.Elements;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var event1 = new EventElement("event1");
                struct1.AddChild(event1);

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

3.2.4 Add a data element

A data element represents a data point.

The “DataElement” class constructor 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 no getter or setter service handler functions are provided the data element uses its internal getter and setter methods. The data element provides the service elements “getdata” and “setdata”.

The “ReadOnlyDataElement” class implements a data element that is read only, i.e. it has only a getter and no setter method. The “WriteOnlyDataElement” class implements a data element that is write only, i.e. it has only a setter method and no getter method.

The data element maintains an internal data cache. A cache timeout value can be provided in the data element constructor. If the cache timeout is not expired the data element getter function returns to value from the cache. If the timeout is expired the getter function is called and the cache is updated. If the setter function updates the value of the data element and the value has changed from the value in the cache a “datachanged” event is raised.

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, array, and object.

3.2.4.1 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 ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.Elements.Formats;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var bool1 = new DataElement<bool>("bool1",
                    GetBool1,
                    SetBool1,
                    format: new BooleanFormat());
                struct1.AddChild(bool1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static bool GetBool1(IDataElement element)
        {
            return _bool1;
        }

        private static void SetBool1(IDataElement element, bool value)
        {
            _bool1 = value;
        }
        private static bool _bool1 = true;
    }
}
```

3.2.4.2 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 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 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var int1 = new DataElement<int>("int1",
                    GetInt1,
                    SetInt1,
                    format: new IntegerFormat(new IntegerValuation(0, 100)));

                struct1.AddChild(int1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static int GetInt1(IDataElement element)
        {
            return _int1;
        }

        private static void SetInt1(IDataElement element, int value)
        {
            _int1 = value;
        }
        private static int _int1 = 10;
    }
}
```

3.2.4.3 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 ifmIoTCore;
    using ifmIoTCore.Common;
    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 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var float1 = new DataElement<float>("float1",
                    GetFloat1,
                    SetFloat1,
                    format: new FloatFormat(new FloatValuation(-9.9f, 9.9f, 3)));
                struct1.AddChild(float1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static float GetFloat1(IDataElement element)
```

```
        {
            return _float1;
        }

        private static void SetFloat1(IDataElement element, float value)
        {
            if (_float1.EqualsWithPrecision(value)) return;
            _float1 = value;
        }
        private static float _float1 = 1.2345f;
    }
}
```

3.2.4.4 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 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 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var string1 = new DataElement<string>("string1",
                    GetString1,
                    SetString1,
                    format: new StringFormat(new StringValuation(0, 100)));
                struct1.AddChild(string1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static string GetString1(IDataElement element)
        {
            return _string1;
        }

        private static void SetString1(IDataElement element, string value)
        {
            _string1 = value;
        }
        private static string _string1 = "Hallo";
    }
}
```

3.2.4.5 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;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var enum1 = new DataElement<int>("enum1",
                    GetEnum1,
                    SetEnum1,
                    format: new IntegerEnumFormat(new IntegerEnumValuation(
                        new Dictionary<string, string>
                        {
                            {"0", "null"},
                            {"1", "one"},
                            {"2", "two"},
                            {"3", "three"}
                        }, _enum1)));
                struct1.AddChild(enum1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static int GetEnum1(IDataElement element)
        {
            return _enum1;
        }

        private static void SetEnum1(IDataElement element, int value)
        {
            _enum1 = value;
        }
        private static int _enum1 = 1;
    }
}
```

3.2.4.6 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.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 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);
```

```
                var array1 = new DataElement<int[]>("array1",
                    GetArray1,
                    SetArray1,
                    format: new ArrayFormat(new ArrayValuation(Format.Types.Number,
                        new IntegerFormat(new IntegerValuation(0, 100)))));
                struct1.AddChild(array1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }

        private static int[] GetArray1(IDataElement element)
        {
            return _array1;
        }

        private static void SetArray1(IDataElement element, int[] value)
        {
            if (_array1.SequenceEqual(value)) return;
            _array1 = value;
        }
        private static int[] _array1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    }
}
```

3.2.4.7 Add an object data element

To add a new object data element to the IoTCore, you first create the field format objects, which are the members of the data element's format specifier. Then create an object of the class "DataElement" with an object format specifier 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
    {
        static void Main()
        {
            try
            {
                var ioTCore = IoTCoreFactory.Create("MyIoTCore");

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                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 = new DataElement<UserData>("object1",
                    GetObject1,
                    SetObject1,
                    format: new ObjectFormat(new ObjectValuation(new List<Field>
                  {
                      intField,
                      floatField,
                      stringField
                  })));
                struct1.AddChild(object1);
            }
```

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

        private static UserData GetObject1(IDataElement element)
        {
            return _object1;
        }

        private static void SetObject1(IDataElement element, UserData value)
        {
            if (_object1.Equals(value)) return;
            _object1 = value;
        }
        private static UserData _object1 = new UserData();
    }

    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;
        }
    }
}
```

3.3 Getting Elements by Address

To get a specific element from the IoTCore element tree you use the element address.

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

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var dataElement = new DataElement<string>("string1");
                struct1.AddChild(dataElement);

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

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

3.4 Add User Specific Data to Elements

Each element has a member field of type “object” called “UserData”. This field allows you to add a reference to 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 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 ifmIoTCore;
    using ifmIoTCore.Elements;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var service1 = new ActionServiceElement(
                    "service1",
                    HandleService1)
                {
                    UserData = new UserData()
                };

                struct1.AddChild(service1);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();
        }

        private static void HandleService1(IServiceElement element, int? cid)
        {
            var userData = (UserData)element.UserData;
            Console.WriteLine($"Do something with {userData}");
        }
    }

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

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

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

```
        }
    }
}
```

3.5 SUBSCRIBE TO EVENTS

3.5.1 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;
    using ifmIoTCore.Elements.EventArguments;

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

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

                // Create an element and do not raise a treechanged event
                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

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

                // Remove an element and raise a treechanged event
                ioTCore.Root.RemoveChild(struct1, true);
            }
            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;
            }
        }
    }
}
```

3.5.2 Add an event handler to an event element

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

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

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var event1 = new EventElement("event1");
                struct1.AddChild(event1);

                event1.Subscribe(HandleEvent1);

                eventTimer = new Timer(RaiseEvent1, event1, 5000, 60000);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();

            eventTimer?.Dispose();
        }

        private static void HandleEvent1(object sender)
        {
            Console.WriteLine("HandleEvent1 called");

            Console.WriteLine($"Received event from {((IEventElement)sender).Address}");
        }

        private static void RaiseEvent1(object param)
        {
            Console.WriteLine("RaiseEvent1 called");

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

3.5.3 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.Common.Variant;
    using ifmIoTCore.Elements;
```

```
    using ifmIoTCore.Elements.Formats;
    using ifmIoTCore.Elements.ServiceData.Events;
    using ifmIoTCore.Elements.ServiceData.Requests;
    using ifmIoTCore.Elements.Valuations;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var service1 = new SetterServiceElement("service1", HandleService1);
                struct1.AddChild(service1);

                var string1 = new ReadOnlyDataElement<string>("string1",
                    GetString1,
                    format: new StringFormat(new StringValuation(0, 100)));
                struct1.AddChild(string1);

                var event1 = new EventElement("event1");
                struct1.AddChild(event1);

                var data = Variant.FromObject(
                    new SubscribeRequestServiceData("/struct1/service1",
                        new List<string> { string1.Address }));
                event1.SubscribeServiceElement.Invoke(data);

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

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

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

            var eventServiceData = Variant.ToObject<EventServiceData>(data);

            Console.WriteLine($"Event source={eventServiceData.EventAddress}");
            Console.WriteLine($"Event number={eventServiceData.EventNumber}");
            foreach (var (key, value) in eventServiceData.Payload)
            {
                Console.WriteLine($"{key}={value.Code},{value.Data}");
            }
        }
    }
}
```

3.6 RAISE EVENTS

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

```
namespace Sample18
{
    using System;
```

```
    using System.Threading;
    using ifmIoTCore;
    using ifmIoTCore.Elements;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);

                var event1 = new EventElement("event1");
                struct1.AddChild(event1);

                event1.Subscribe(HandleEvent1);

                eventTimer = new Timer(RaiseEvent1, event1, 5000, 60000);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();

            eventTimer?.Dispose();
        }

        private static void HandleEvent1(object sender)
        {
            Console.WriteLine("HandleEvent1 called");

            Console.WriteLine($"Received event from {((IEventElement)sender).Address}");
        }

        private static void RaiseEvent1(object param)
        {
            Console.WriteLine("RaiseEvent1 called");

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

3.7 Create Links to Elements

You can create a link from a source element to a target element. A link creates an entry in the source element which references the target element. The target element is then accessible via its original address and via its linked address. You can also give the link a meaningful, user friendly name.

```
namespace Sample10
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.Elements;

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

                var struct1 = new StructureElement("struct1");
                ioTCore.Root.AddChild(struct1);
                var struct2 = new StructureElement("struct2");
```

```
                struct1.AddChild(struct2);
                var data1 = new DataElement<int>("data1", value:123);
                struct2.AddChild(data1);

                // Create a link from root element to data1 element
                ioTCore.Root.AddLink(data1, "link_data1");

                // Get the element via link
                var element = (IDataElement)ioTCore.GetElementByAddress("/link_data1");
                Console.WriteLine(element.Value);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
}
```

3.8 Create Network Adapters

3.8.1 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.MessageConverter.Json.Newtonsoft;
    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 MessageConverter());

                ioTCore.RegisterServerNetAdapter(httpServer);

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

3.8.2 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.MessageConverter.Json.Newtonsoft;
    using ifmIoTCore.NetAdapter.Http;

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

                ioTCore.RegisterClientNetAdapterFactory(
                    new HttpClientNetAdapterFactory(
                        new MessageConverter()));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

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

3.8.3 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 Sample28
{
    using System.Net;
    using System.Threading;
    using ifmIoTCore;
    using ifmIoTCore.Elements;
    using ifmIoTCore.MessageConverter.Json.Newtonsoft;
    using ifmIoTCore.NetAdapter.Mqtt;

    class Program
    {
        static void Main()
        {
            using (var manualResetEvent = new ManualResetEventSlim())
            using (var iotCore = IoTCoreFactory.Create("id"))
            using (var mqttServerNetAdapter = new MqttServerNetAdapter(iotCore, iotCore.Root,
new MessageConverter(), new IPEndPoint(IPAddress.Loopback, 1883)))
            using (var mqttClientNetAdapterFactory = new MqttNetAdapterClientFactory(new Mes-
sageConverter()))
            {
                iotCore.RegisterClientNetAdapterFactory(mqttClientNetAdapterFactory);

                mqttServerNetAdapter.RequestMessageReceived += (s, e) =>
                {
                    e.ResponseMessage = iotCore.HandleRequest(e.RequestMessage);
                };

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

                iotCore.RegisterServerNetAdapter(mqttServerNetAdapter);
```

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

                iotCore.Root.AddChild(serviceElement);

                mqttServerNetAdapter.Start();

                manualResetEvent.Wait();

                mqttServerNetAdapter.Stop();
            }
        }
    }
}
```

3.8.3.1 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"
                            }
                          ]
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
```

3.8.4 Create a mqtt client network adapter factory

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

```
namespace Sample29
{
    using System;
    using ifmIoTCore;
    using ifmIoTCore.MessageConverter.Json.Newtonsoft;
    using ifmIoTCore.NetAdapter.Mqtt;

    class Program
    {
        static void Main()
        {
            try
            {
                var iotCore = IoTCoreFactory.Create("test");
                iotCore.RegisterClientNetAdapterFactory(new MqttNetAdapterClientFactory(new
MessageConverter()));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();

        }
    }
}
```

3.8.4.1 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.

3.8.4.2 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:

```
namespace Sample30
{
    using System;
    using System.Collections.Generic;
    using System.Net;
    using ifmIoTCore.Common.Variant;
    using ifmIoTCore.Elements.ServiceData;
    using ifmIoTCore.Elements.ServiceData.Events;
    using ifmIoTCore.MessageConverter.Json.Newtonsoft;
    using ifmIoTCore.Messages;
    using ifmIoTCore.NetAdapter;
    using ifmIoTCore.NetAdapter.Mqtt;

    class Program
    {
        static void Main()
        {
            IClientNetAdapter client = new MqttClientNetAdapter(
                "cmdTopic",
                new IPEndPoint(
                    IPAddress.Parse("192.168.1.1"),
                    1883),
                new MessageConverter(),
                TimeSpan.FromSeconds(1));


            var eventData = new EventServiceData(0, "/identifier",
                new Dictionary<string, CodeDataPair>
                {
                    {
                        "/identifier",
                        new CodeDataPair(ResponseCodes.Success, Variant.FromObject(3))
                    }
                });

            client.SendEvent(new Message(10, 0, "/serviceHandler", Variant.FromObject(event-
Data)));
        }
    }
}
```

3.8.5 The OPC UA server Network Adapter

3.8.5.1 Description

The OPC UA Server network adapter provides an integrated OPC UA Server and maps the elements from the IoTCore element tree to corresponding nodes in the OPC UA Server address space. Each node in the OPC UA Server is linked to its counterpart element in the IoTCore, thus making all the functionality of the IoTCore elements available through the OPC UA Server.

The network adapter provides two different representations of the IoTCore:

The first one maps the IoTCore elements and the element tree structure to nodes following the IoTCore specification. This means, that the nodes and the nodes structure in the address space directly reflect the elements and the element structure of the IoTCore. The nodes have the same identifiers, type, format, data, and semantic like the elements in the IoTCore. All elements in the IoTCore are mapped as nodes and the hirarchy of the IoTCore element tree is also preserved in the hirarchy of the nodes in the OPC UA server. If the IoTCore represents IO-Link devices and IODDs for those devices are available in the network adapter's IODD directory the device nodes are extended with nodes generated from the IODD.

The other representation maps only IO-Link related elements of the IoTCore to nodes in the OPC UA Server according to the OPC UA companion specification for IO-Link. This specification defines how IO-Link master, port, and device with and without IODD are represented in the OPC UA Server.

3.8.5.2 OPC UA Server configuration

The OPC UA Server network adapter can be configured from the IoTCore configuration file. This configuration file is called "DataStore.json" and must be placed in the "%ProgramData%/ifm/IoTCore" directory. On Windows this typically is "C:\ProgramData\ifm\IoTCore". On Linux this typically is "/usr/share/ifm/IoTCore". In this configuration file is a JSON object for the OPC UA Server network adapter that controls the basic features and settings of the network adapter:

```
"OpcUa": {

  "UseIodd": true,
  "IoddDirectory": "%ProgramData%/ifm/Iodds",
  "StartIoLinkMapper": true,
  "AcceptUntrustedCertificate": true
}
```

3.8.5.3 Create and install SSL server certificates

SSL server certificates can be installed in the server's certificate directory "%ProgramData%/ifm/IoTCore/OpcUa/pki/own". The certificate must be placed in the "certs" subdirectory. The companion private key goes in the "private" subdirectory. The certificate must be in "der" format, the private key in "pfx" format. If no certificate is installed the server will create its own self-signed certificate at first start.

3.8.5.4 IoTCore element mapping according to IoTCore specification

Reading and writing IoTCore data elements through OPC UA variable nodes

IoTCore data elements are directly mapped to OPC UA variable nodes. Reading and writing OPC UA variables is directly mapped to calling the "getdata" and "setdata" services of the corresponding data element in the IoTCore.

Calling IoTCore services through OPC UA method nodes

IoTCore service elements are directly mapped to OPC UA method nodes. Calling a method is directly mapped to calling the service of the corresponding service element in the IoTCore. If a method requires input parameters the input parameters are provided as JSON string in the input argument parameter of the method. The JSON string is exactly the "data" part of the IoTCore service request message. If, for instance, a "setdata" service is called through a JSON request message via HTTP the message looks like this:

```
{
  "code":10,
  "cid": 10,
  "adr": "/struct1/string1/setdata",
  "data": { "newvalue" : "huhu" }
}
```

The same service called through the corresponding OPC UA method node the input argument would be just:

{ "newvalue" : "huhu" }

IODD support

IODD support in the OPC UA Server network adapter must be enabled in the IoTCore configuration file by setting the "UseIodd" flag to true. The IODD files for the devices must be placed in the directory which is specified in the "IoddDirectory" variable in the IoTCore configuration file. If the IoTCore represents IO-Link devices and IODDs for those devices are available in the network adapter's IODD directory the device nodes are extended with nodes generated from the IODD.

3.8.5.5 IoTCore element mapping according to OPC UA companion specification for IO-Link

This representation of the IoTCore's IO-Link related elements is enabled by setting the variable "StartIoLinkMapper" in the IoTCore configuration file to true. If the IoTCore represents IO-Link related devices (IO-Link master, port, device) nodes for those devices are created in accordance with the OPC UA companion specification for IO-Link.

3.8.5.6 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.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();
        }
    }
}
```

3.8.6 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 (**3.8.8.1**) 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();
        }
    }
}
```

3.8.7 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();
        }
    }
}
```

3.8.8 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();
        }
    }
}
```

3.8.8.1 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.

3.9 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();
        }
    }
}
```

3.10 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.

3.11 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
    }
}
```

4 DATA REPRESENTATION - VARIANT

The classes that represent the data of services and dataelements that users of the iotcore framework will create cannot be inferred by the iotcore framework. So there is a need to represent these data classes dynamically. This leads to an abstract representation of data that we describe with the following classes:

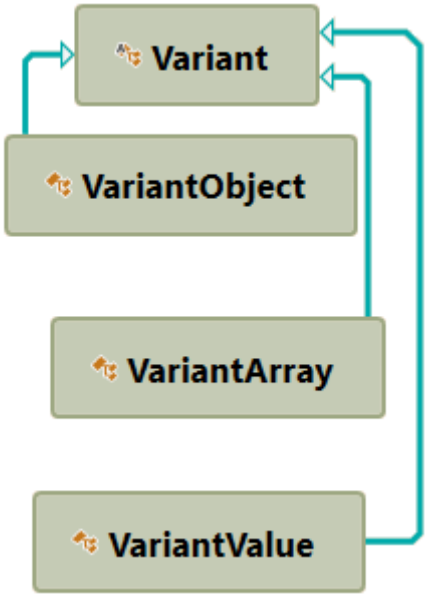


Figure 1 Inheritance Diagram

Variant – Baseclass

VariantValue – Simle Value type (like for instance: int, string, byte, char …)

VariantArray – An Array of all other types

VariantObject – A dictionary of key-value pairs.

With this abstraction common objects can be described:

```
namespace Sample31
{
    using System;
    using ifmIoTCore.Common.Variant;

    class Program
    {
        static void Main()
        {
            Variant variantValue1 = new VariantValue(1);
            Variant variantValue2 = Variant.FromObject(2);
            Variant variantValue3 = Variant.FromObject(3.0f);
            Variant variantValue4 = Variant.FromObject('c');

            Variant array = new VariantArray
            {
                variantValue1,
                variantValue2,
                variantValue3,
                variantValue4
```

```
            };

            Variant fruits = new VariantArray(new string[] { "apple", "banana", "cherry" });

            Variant obj = new VariantObject
            {
                { "items", array },
                { "fruits", fruits }
            };

            WorkWithVariant(array);
            WorkWithVariant(fruits);
            WorkWithVariant(obj);
        }

        static void WorkWithVariant(Variant variant)
        {
            if (variant is VariantObject variantObject)
            {
                var numbers = variantObject["items"];
                var firstItemAsInt = numbers.AsVariantArray()[0].ToObject<int>();
            }
            else if (variant is VariantArray variantArray)
            {
                var firstItem = variantArray[0];

                try
                {
                    int firstItemAsInt = firstItem.ToObject<int>();
                }
                catch (FormatException e)
                {
                    // This exception will be thrown when working with the fruits array, since
the string "apple" is not convertible to an integer.
                }


                // This will succeed with both the arrays, the results are "apple" and "1".
                string firstItemAsString = firstItem.ToObject<string>();

                // The cast operators are overloaded for VariantValue so casts to simple types
also do a conversion.
                string firstItemAsStringCast = (string)(VariantValue)firstItem;
            }
        }
    }
}
```

The Variant class has two static methods, that allow own classes to be converted to instances of variant.

```
public static Variant FromObject(object data)
```

```
public static T ToObject<T>(Variant data)
```

There are restrictions on the type `T` that should be converted to and from Variant:

- The type must have a public parameterless constructor
- The conversion does not support object references

5 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”.