Migration Guide to dataFEED OPC UA .NET Standard SDK 2.60

by Softing Industrial Automation GmbH

This is the migration guide for dataFEED OPC UA .NET SDK to the new dataFEED OPC UA .NET Standard SDK.

Table of Contents

1 Welcome

optimize! softing

dataFEED OPC UA .NET Standard SDK enable fast integration of the OPC Unified Architecture standard communication interface within end user applications. It provides a comprehensive set of source code and binaries covering OPC UA functionality for building Client respective Server interface thus allowing short time to market of your solutions.

The SDK's libraries are accompanied with an easy to use client side API, allowing the user to concentrate on the own business requirements implementation of his product and hiding the OPC UA protocol complexity.

- **Leading edge easy-to-use client interface**
- **Suitable for a wide range of applications**
- **Higher portability**
- **Validated by solid, well established battery of tests**

The dataFEED OPC UA .NET Standard SDK includes:

- OPC UA dataFEED OPC UA .NET Standard SDK Client Library & dataFEED OPC UA .NET Standard SDK Server Library - an easy to use API built on top of the OPC UA .NET Standard Stack from OPC Foundation.
- Unchanged access to OPC UA .NET Standard Stack from OPC Foundation.
- Reworked, comprehensive documentation focusing on the user's needs on different level of OPC UA expertise.
- Support for custom complex data types decoding, compliant with OPC UA 1.03 specification.
- Sample server and client applications.

The reader is expected to be familiar with Microsoft .NET and C# terminology as well as the basic OPC UA concepts. However the documentation will try to fill some gaps between the UA concepts and the way they are realized within the source code.

This document does not describe every class in the OPC UA .NET Standard Development Kit. It is intended to supplement the Visual Studio IntelliSense documentation and help developers understand how to develop production servers and clients.

1.1 What's New

A list of the new features added in the released versions is provided below:

Version 2.60

- **dataFEED OPC UA .NET Standard SDK Server Library:**

- Support for reverse connectivity.
- In *NodeManager* class:
    - The *CreateObjectFromType()* and *CreateVariableFromType()* methods were enhanced with an additional parameter (*createOptionalProperties*) that specifies if the properties that have optional modelling rule are instantiated when creating an object/variable. Note that the types from OPC UA .NET Standard Stack from OPC Foundation will not create optional properties.
    - The *ReportEvent* method was enhanced to accept also the node id of the event type that will be reported.
- Bug fixes.

- **dataFEED OPC UA .NET Standard SDK Client Library:**

- Support for reverse connectivity.
- Asynchronously methods added for Connect (*ConnectAsync*), Disconnect (*DisconnectAsync*), GetEndpoints (*GetEndpointsAsync*), DiscoverServers (*DiscoverServersAsync*) and DiscoverServersOnNetwork (*DiscoverServersOnNetworkAsync*).
- Bug fixes.

- **dataFEED OPC UA .NET Standard SDK PubSub Library:**

- Bug fixes.

SDK libraries are compiled using NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.363.49. When using dataFEED OPC UA .NET Standard SDK version 2.60 it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

1.2 Revision History

Version 2.50

- **dataFEED OPC UA .NET Standard SDK Server Library:**

  - Internal support for AccessRestrictions, RolePermissions and UserRolePermissions attributes.
  - Support for GDS and Push Management of application certificates and trust lists.
  - Bug fixes.

- **dataFEED OPC UA .NET Standard SDK Client Library:**

  - Support for GDS and Pull Management of application certificates and trust lists.
  - ReadNode method from *ClientSession* will not throw a ServiceResultException when the node is not found, but will return an instance of *VariableNodeEx*. Status codes for each attribute are available via *GetStatusCode()* method.
  - Bug fixes.

- **dataFEED OPC UA .NET Standard SDK PubSub Library:**

  - Bug fixes.

- .NET Framework support updated to version **4.6.2** in accordance with *OPC UA .NET Standard Stack from OPC Foundation* .

- The support for **Visual Studio 2015** was removed.

SDK libraries are compiled using NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.362.42. When using dataFEED OPC UA .NET Standard SDK version 2.50 it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

Version 2.40

- **dataFEED OPC UA .NET Standard SDK Server Library:**

  - Support for creating custom *DataType*, *VariableType* and *ObjectType* nodes and instances
  - Extended support for complex data types:
    - Support for custom *Structure*, *StructureWithOptionalFields*, *Union*, *Enumeration* and *OptionSet* types.
    - Expose *DataTypeDefinition* attribute of DataType nodes to allow clients to read custom data types.
    - Enhanced ImportNodeSet functionality for reading the *DataTypeDefinition* attribute of DataType nodes.
    - Sample code for creating and handling custom types.
  - Support for creating custom variable and object types.
  - Bug fixes.
  - .NET Standard 1.4 support removed.

- **dataFEED OPC UA .NET Standard SDK Client Library:**

- Extended support for complex data types:
  - Support for custom *Structure*, *StructureWithOptionalFields*, *Union*, *Enumeration* and *OptionSet* types.
  - Load server custom data types by reading the *DataTypeDefinition* attribute of DataType nodes for V1.04 servers.
    - Old setting DecodeCustomDataTypes changed its semantic and it configures if the custom data types information is loaded from *DataTypeDefinition* attribute.
    - New setting DecodeDataTypeDictionaries has the semantics of old *DecodeCustomDataTypes* setting and configures if the custom data types information is loaded from data type dictionaries.
  - Sample code for handling custom data types.
- Bug fixes.
- .NET Standard 1.4 support removed.

- **dataFEED OPC UA .NET Standard SDK PubSub Library:**

- Bug fixes.

SDK libraries are compiled using NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.359.31. When using dataFEED OPC UA .NET Standard SDK version 2.40 it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

Version 2.30

- **dataFEED OPC UA .NET Standard SDK PubSub Library:**

- Provides simplified API for configuring a PubSub application.
- SampleServer provides a new dedicated node manager that uses dataFEED OPC UA .NET Standard SDK PubSub Library and behaves like a Publisher and Subscriber.
- Changes in licensing namespaces:

In version 2.20 the *PubSub* licensing classes were part of *Softing.Opc.Ua.**Private*** namespace. Starting with version 2.30 the *PubSub* licensing classes are part of *Softing.Opc.Ua.**PubSub*** namespace. Please change your code to point to the new namespace.

- **dataFEED OPC UA .NET Standard SDK Server Library:**

- Bug fixes.
- Changes in licensing namespaces:

In version 2.20 the *Server* licensing classes were part of *Softing.Opc.Ua.**Private*** namespace. Starting with version 2.30 the *Server* licensing classes are part of *Softing.Opc.Ua.**Server*** namespace. Please change your code to point to the new namespace.

- **dataFEED OPC UA .NET Standard SDK Client Library:**

- Bug fixes.
- Changes in licensing namespaces:

In version 2.20 the *Client* licensing classes were part of *Softing.Opc.Ua.**Private*** namespace. Starting with version

2.30 the *Client* licensing classes are part of *Softing.Opc.Ua.**Client*** namespace. Please change your code to point to the new namespace.

SDK libraries are compiled using NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.357.28. When using dataFEED OPC UA .NET Standard SDK version 2.30 it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

Version 2.20

- The initial version of **dataFEED OPC UA .NET Standard SDK PubSub Library:**

  - Provides an API for simplified OPC UA Publisher and Subscriber development using UDP transport and binary encoded messages.
  - Provides SamplePublisher and SampleSubscriber projects that use the simplified API from dataFEED OPC UA .NET Standard SDK PubSub Library.

- **dataFEED OPC UA .NET Standard SDK Server Library:**

  - Enhanced ImportNodeSet functionality that allows instantiating data types loaded from dictionaries.
  - New method *GetDefaultValueForDatatype* was added to *NodeManager*. It creates and returns the default value for the specified data type, including data types imported from NodeSet2 XML files.
  - Added support for **Aes256_Sha256_RsaPss** security policy.

- **dataFEED OPC UA .NET Standard SDK Client Library:**

  - *ClientSession.KeepAlive* handling was moved on a separate thread to avoid possible deadlocks.

SDK libraries are compiled using NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.355.26. When using dataFEED OPC UA .NET Standard SDK version 2.20 it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

Version 2.10

- **"Softing OPC UA .NET Standard Toolkit"** was renamed to **"dataFEED OPC UA .NET Standard SDK".**

- OPC UA Server Compliance Certification.

- dataFEED OPC UA .NET Standard SDK Server Library:

  - Enhanced ImportNodeSet functionality that handles duplicate nodes.
  - Added Export NodeSet functionality.
  - Added create instance from OPC UA type specified by *NodeId* functionality.
  - Added sample code for file transfer.

- Added sample code for temporary file transfer.

- Added support for **Aes128_Sha256_RsaOaep** security policy.

- Removed **Windows 8.1** support due to known security limitations incompatible with dataFEED OPC UA .NET Standard SDK implementation.

- Properties marked as deprecated in ServerToolkitConfiguration.

SDK libraries are compiled using NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.3.354.23. When using dataFEED OPC UA .NET Standard SDK version 2.10 it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

Version 2.00

Version 2.00 provides the following features for **Softing OPC UA .NET Standard Client Toolkit** (the product was renamed to **"dataFEED OPC UA .NET Standard SDK Client Library"** in version 2.10):

- Dll name was changed from *Softing.Opc.Ua.dll* to *Softing.Opc.Ua.Client.dll.*

Namespace changes:

  - All entities from namespace *Softing.Opc.Ua* were moved to namespace *Softing.Opc.Ua.Client* (*SecurityPolicy* enum, *SDK Utils* and *UaApplication* classes)
  - All entities from namespace *Softing.Opc.Ua.Private* were moved to *Softing.Opc.Ua.Client.Private* (*License* class and *LicenseFeature* enum)
  - All entities from namespace Softing.Opc.Ua.Types were moved to Softing.Opc.Ua.Client.Types.

- *ApplicationConfigurationEx* class available in version 1.00 and 1.10 in namespace *Softing.Opc.Ua* was removed. We are using the *ApplicationConfiguration* class available in OPC UA .NET Standard Stack from OPC Foundation . A new configuration class *ClientToolkitConfiguration* was added to the Softing OPC UA .NET Standard Client Toolkit, it is used as a configuration extension as described in Client configuration section.
- New method added to *ClientSubscription* class. *ConditionRefresh* method tells the server to refresh all conditions (alarms) being monitored by the subscription.

Version 2.00 provides the initial version of **Softing OPC UA .NET Standard Server Toolkit:**

- Provides an API for simplified server development. The dataFEED OPC UA .NET Standard SDK Server Library offers a simplified approach for Data Access, Methods and Standard Event Subscriptions
- SampleServer and XamarinSampleServer are refactored to use the new Softing OPC UA .NET Standard Server Toolkit (Softing.Opc.Ua.Server.dll).

SDK libraries are compiled using NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.353.15. When

using dataFEED OPC UA .NET Standard SDK version 2.00 it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

Version 1.10

Version 1.10 provides the following features for **Softing OPC UA .NET Standard Client Toolkit:**

- Support for Linux distributions: Ubuntu 16.04 and Debian 9.30.
- Support for Android. Sample Client and Sample Server for Android implemented with Xamarin Forms were added in the samples folder.
- Softing OPC UA .NET Standard Client Toolkit is compiled using NuGet package *OPCFoundation.NetStandard.Opc.Ua* - version 1.3.352.10. When using version 1.10 of the Softing OPC UA .NET Standard Toolkit it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

Version 1.00

Initial release of **Softing OPC UA .NET Standard Toolkit** provides only **Softing OPC UA .NET Standard Client Toolkit** functionality.

Note: The product was renamed to **"dataFEED OPC UA .NET Standard SDK"** in version 2.10.

New license keys. The old license keys (valid for dataFEED OPC UA .NET SDK 1.4x) are not valid for the new version anymore.

The known easy client API (familiar to dataFEED OPC UA .NET SDK 1.4x users) ported to the .NET Standard Stack. Unfortunately it required refactoring of the API interface itself, therefore the backward compatibility (at code level) is broken. We addressed that by not making it too diverse compared to the previous one and by offering within this documentation a migration guide.

Windows support only. This is NOT meaning the Softing OPC UA .NET Standard Toolkit cannot compile and run on other platforms supported by .NET Standard; it means for the current version the tests were performed only on Windows OS.

Discontinued support for client side redundancy. It will be refactored and completed in the next versions.

For those familiar with the support of the complex data types decoding in the dataFEED OPC UA .NET SDK 1.4x we offer a continuation of it as we observed that many integrators appreciated it.

Softing OPC UA .NET Standard Client Toolkit is compiled using NuGet package *OPCFoundation.NetStandard.Opc.Ua* - version 1.3.350.4. When using version 1.00 of the Softing OPC UA .NET Standard Toolkit it is important to use the same version of OPC UA .NET Standard Stack from OPC Foundation.

1.3 System Requirements

The version 2.60 of the dataFEED OPC UA .NET Standard SDK requires one of the following supported operating systems:

- Windows 7 SP1
- Windows 10
- Windows Server 2012 SP1
- Windows Server 2012 R2
- Windows Server 2016
- Debian 9.30 (no development support, only runtime)
- Ubuntu 16.04 (no development support, only runtime)

The development package requires support from Microsoft .NET Standard 2.0 and .NET Core 2.0.

The installation deploys projects and solutions for Microsoft Visual Studio IDE, therefore at least one of the following versions should be installed:

IDE version	Prerequisites
Visual Studio 2017	.NET Core 2.0 SDK - https://dotnet.microsoft.com/download/dotnet-core/2.0
	.NET 4.6.2 SDK - https://dotnet.microsoft.com/download/dotnet-framework/net462

1.4 OPC UA Supported Features

Communication protocols

- opc.tcp - binary encoding

Server

- Data Access Server
- Method Server
- DataChange Subscription Server
- Standard Event Subscription Server
- Address Space Notifier Server
- Standard UA Server
- Local Discovery Server
- Core Server
- History Data Access Server
- Alarms and Conditions Server

Client

- Event Subscriber Client
- DataAccess Client
- Method Client
- DataChange Subscriber Client
- AddressSpace Lookup Client
- Discovery Client
- Core Client
- History Data Access Client
- Alarms and Conditions Client

Unsupported Features

- TransferSubscriptions Service
- Cancel Service
- NodeManagement Service Set
  - AddNodes Service
  - AddReferences Service
  - DeleteNodes Service
  - DeleteReferences Service
- Query Service Set
  - QueryFirst Service
  - QueryNext Service

2 Client API Migration Guide

dataFEED OPC UA .NET Standard SDK introduces breaking changes for previous versions of OPC UA SDK.

In **old SDK version** most of the stack classes had a corresponding wrapped class with the same name. In **dataFEED OPC UA .NET Standard SDK** we kept only those wrapped classes that added functionality and they have the stack name suffixed by *Ex* from *Extension (e.g. ReferenceDescriptionEx).* Old code that needs to be migrated must rename those classes that have been extended in **dataFEED OPC UA .NET Standard SDK** to their new names and point to stack namespaces for those classes that do not have a wrapper anymore.

Note: Each application that uses dataFEED OPC UA .NET Standard SDK version 2.60 must add reference to NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.363.49. The nuget package contains the OPC UA stack implementation maintained by OPC Foundation under GIT repository.

In **dataFEED OPC UA .NET Standard SDK** we added *Client* prefix to *Session, Subscription, MonitoredItem* and *Method* classes to be able to easily differentiate between those and the ones provided by the stack. Therefore there are *ClientSession*, *ClientSubscription*, *ClientMonitoredItem* and *ClientMethod* classes.

General

1. Copy your solution folder to a new location.
2. Open copy solution in Visual Studio 2017.
3. Change target framework to .Net Framework 4.6.2. in project properties window (required by .net standard 2.0 libraries).
4. Remove reference to old *Softing.Opc.Ua.Toolkit.dll* from your project.
5. Add reference to Softing.Opc.Ua.Client.dll to your project.
6. Add reference to NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.363.49.
7. Rename namespaces as described in table below - section **1**.
8. Create application configuration. You can refactor code like in table below - section **2.1,** or you can create a configuration file similar to the one provided with SampleClient application, rename it, change it for your needs and use it. **Please do not change the order of XML elements in configuration file.**
9. Apply **Application** refactor explained in table below - section **3.**
10. Use *ClientSession* from *Softing.Opc.Ua.Client* instead *of Softing.Opc.Ua.Toolkit.Client.Session.* See table below - section **5.**
11. Instantiate *ClientSession with uaApplication.CreateSession* like described in section **5.1** (table). If there is user authentication logic please use table - section **5.2** for refactor details.

Discovery functionality

1. Apply all that applies from **General.**
2. Apply refactor from table section **4** (**4.1**, **4.2**).

Connect Functionality

1. Apply all that applies from **General.**
2. Apply refactor from table section **5** (**5.1**, **5.2**).

Browse functionality

1. Apply all that applies from **General.**
2. Remove handling of *Session.ContinuationPointReached* event, see table below section **7.1.**
3. Use *BrowseDescriptionEx* instead of *BrowseOptions* (table - sections **7.1**, **7.2**).
4. Use *ReferenceDescriptionEx* instead of *ReferenceDescription* for *Browse* methods return *(*table section **7.4**).
5. To transform *ExpandedNodeId* to *NodeId* for *Browse* methods use refactor from table - section **8.**

TranslateBrowsePath functionality
1. Apply all that applies from **General.**
2. Rename *BrowsePath* to *BrowsePathEx (*table section **7.5**).
3. Rename *BrowsePathResult* to *BrowsePathResultEx (*table section **7.6**).

Data Access
1. Apply all that applies from **General.**
2. Rename *Subscription* class to *ClientSubscription (*table section **6**).
3. Rename *MonitoredItem* to *ClientMonitoredItem* (table sections **9, 9.1, 10**).
4. Replace *AttributeId with Attributes* (table section **10**).

Read complex values or enum values
1. Apply all that applies from **Data Access.**
2. Ensure that type dictionaries are loaded in current *ClientSession* and then use *DataValueEx.ProcessedValue* field to retrieve complex value/enum value. Complex values are instances of *StructuredValue* and enum types are instances of *EnumValue. They are* defined in Softing.Opc.Ua.Client.dll as opposed to old SDK version where *StructuredValue and EnumValue* types were part of the stack.

Table section 12.1 describes the approach.

Write complex values
1. Apply all that applies from **Data Access.**
2. Ensure that type dictionaries are loaded in current *ClientSession*
3. Check sample code *ReadWriteClient.WriteComplexValue()* for a concrete example of how to write a complex value. The example first reads the type information for the node id, then gets default value for that type and ten changes the default value to desired value and writes it to the node. See table **section 12.2**.

Monitored item
1. Apply all that applies from **Data Access.**
2. Use *EventFilterEx* instead of stack *EventFilter* and *SelectOperandEx* instead of stack *SelectOperand*.

Handling alarms
1. Apply all that applies from **Data Access.**
2. Use *EventFilterEx* instead of *EventFilter* to specify the event filter for alarms. See table section **9.2.**

History read
1. Apply all that applies from **General.**
2. Remove *HistoryContinuationPointReached* event handling from session. *ClientSession* implememntation does not provide this event anymore. (see table section **5.3**).
3. **Read raw** changes are described in table section **13.1.**
4. **Read at time** changes are described in table section **13.2.**
5. **Read processed** changes are described in table section **13.3.**

Method call - see table section **14**.
Logging - see table section **15**.

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
1	**Namespaces**	
1.1	Softing.Opc.Ua.Toolkit	remove
1.2	Softing.Opc.Ua.Toolkit.Client	Softing.Opc.Ua.Client
1.3	Softing.Opc.Ua.Toolkit.Client.Nodes	Softing.Opc.Ua.Client.Nodes
1.4	Softing.Opc.Ua.Sdk.* (old Softing OPC Ua SDK)	Opc.Ua.*
2	**Initialize configuration**	
2.1	`Application.Configuration.ApplicationName = "SoftingOpcUaBrowseClient";` `Application.Configuration.ProductUri = "http://industrial.softing.com/OpcUaNetToolkit/BrowseClient";` `// security configuration.` `string applicationFolder = Path.Combine(Assembly.GetExecutingAssembly().Location, @"..\..\..\..\..\..");` `applicationFolder = Path.GetFullPath(applicationFolder);` `Application.Configuration.Security.ApplicationCertificateStore = Path.Combine(applicationFolder, @"c:\tmp\pki\own");` `Application.Configuration.Security.ApplicationCertificateSubject = Application.Configuration.ApplicationName;` `Application.Configuration.Security.TrustedCertificateStore = Path.Combine(applicationFolder, @"c:\tmp\pki\trusted");` `Application.Configuration.Security.TrustedIssuerCertificateStore = Path.Combine(applicationFolder, @"c:\tmp\pki\issuer");` `Application.Configuration.Security.RejectedCertificateStore = Path.Combine(applicationFolder, @"c:\tmp\pki\rejected");` `Application.CertificateValidation += Application_CertificateValidation;` `// The Validate() method ensures that the specified configuration is valid.` `try` `{`	`// Create the ApplicationConfiguration object` `ApplicationConfiguration configuration = new ApplicationConfiguration();` `configuration.ApplicationName = "SoftingOpcUaBrowseClient";` `configuration.ProductUri = "http://industrial.softing.com/OpcUaNetToolkit/BrowseClient";` `// Create new instance of ClientToolkitConfiguration and set desired properties` `ClientToolkitConfiguration clientTkCfg = new ClientToolkitConfiguration();` `clientTkCfg.DiscoveryOperationTimeout = 10000;` `// Set ClientToolkitConfiguration instance as an extension of current ApplicationConfiguration object` `configuration.UpdateExtension<ClientToolkitConfiguration>(new XmlQualifiedName("ClientToolkitConfiguration"), clientTkCfg);` `// security configuration.` `string applicationFolder = Path.Combine(Assembly.GetExecutingAssembly().Location, @"..\..\..\..\..\..");` `applicationFolder = Path.GetFullPath(applicationFolder);` `configuration.SecurityConfiguration = new SecurityConfiguration` `{` `  ApplicationCertificate = new CertificateIdentifier` `  {` `  SubjectName =`

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
	<pre> Application.Configuration.Validate(); } catch (Exception ex) { Console.WriteLine(ex.Message); return false; } // trace configuration Application.Configuration.Trace.LogFileName = @"Logs\BrowseClient.txt"; Application.Configuration.Trace.LogFileMaxSize = 10; Application.Configuration.Trace.LogFileMaxRollBackups = 5; Application.Configuration.Trace.LogFileTraceLevel = TraceLevels.Warning; //enable all masks Application.Configuration.Trace.LogFileTraceMask = 0x00FF00FF; Application.Configuration.Trace.Tracelevel = TraceLevels.Warning; //enable all masks Application.Configuration.Trace.TraceMask = 0x00FF00FF;</pre>	<pre>configuration.ApplicationName, StoreType = CertificateStoreType.Directory, StorePath = @"c:\tmp\pki\own" }, TrustedPeerCertificates = new CertificateTrustList { StoreType = CertificateStoreType.Directory, StorePath = @"c:\tmp\pki\trusted", }, TrustedIssuerCertificates = new CertificateTrustList { StoreType = CertificateStoreType.Directory, StorePath = @"c:\tmp\pki\issuer", }, RejectedCertificateStore = new CertificateTrustList { StoreType = CertificateStoreType.Directory, StorePath = @"c:\tmp\pki\rejected", }, AutoAcceptUntrustedCertificates = true }; // trace configuration configuration.TraceConfiguration = new TraceConfiguration() { OutputFilePath = @"Logs\BrowseClient.txt", DeleteOnLoad = true, TraceMasks = 519 };</pre> *Code snippet is from ConnectClient sample.*
3	**Application**	
3.1	***Application*** class was used to instantiate and configure an OPC Ua client application. <pre>Application.Configuration.ApplicationName = "UA Sample Client"; Application.Configuration.ProductUri = "http://industrial.softing.com/OpcUaNetToolkit/BrowseClient"; //configure the application by code ...</pre>	***UaApplication*** class replaces *Application* class. *UaApplication* is not static class, the user can instantiate and configure multiple *UaApplications* as opposed to old *Application* approach. <pre>// Create the UaApplication object from config file UaApplication application = UaApplication.Create(Constants.ConfigurationFile).Result;</pre> *Code snippet is from Program.cs in sample project.*

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
		Or `// Create the ApplicationConfiguration object` `ApplicationConfiguration configuration = new ApplicationConfiguration();` `configuration.ApplicationName = "UA Sample Client";` `//configure the application by code` `...` `// Create the UaApplication object from config object` `UaApplication application = UaApplication.Create(configuration).Result;` *Code snippet is from ConnectClient sample.*
3.2	***Application.CertificateValidation event*** `Application.CertificateValidation += Application_CertificateValidation;` `private static void Application_CertificateValidation(object sender, CertificateValidationEventArgs e)` `{` `  // Add custom logic for validating the server certificate.` `  // Accept this certificate during the runtime of the application.` `  e.ValidationOption = CertificateValidationOption.AcceptOnce;` `}`	***CertificateValidation event*** *was not created into UaApplication class.* You must use *CertificateValidator.CertificateValidation* event instead. `application.Configuration.CertificateValidator.CertificateValidation += Application_CertificateValidation;` `private static void Application_CertificateValidation(object sender, CertificateValidationEventArgs e)` `{` `  // Add custom logic for validating the server certificate.` `  // Accept this certificate during the runtime of the application.` `  e.Accept = true;` `}`
4	***Discovery***	
4.1	***Application.DiscoverServers****(discoveryUrl)* method used to return a list of *EndpointDescription* objects for all servers found at *discoveryUrl*. `// The method will return all the available server endpoints from the specified machine` `IList<EndpointDescription> endpoints = Application.DiscoverServers(discoveryUrl);` *To get the endpoints you need to call static Application.GetEndpoints(serverDiscoveryUrl)* returns a list	***UaApplication.DiscoverServers****(discoveryUrl)* returns a list of *ApplicationDescription* objects, one for each server found on address. `// the method will return all the registered server applications from the specified machine.` `IList<ApplicationDescription> appDescriptions = uaApplication.DiscoverServers(discoveryUrl);` *To get the endpoints you need to call UaApplication.GetEndpoints(serverDiscoveryUrl)* returns a list of *EndpointDescriptionEx* objects, one for each endpoint found on address.

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
	of *EndpointDescription* objects, one for each endpoint found on address. `// retrieve available endpoints for specified serverDiscoveryUrl.` `IList<EndpointDescription> endpoints = Application.GetEndpoints(serverDiscoveryUrl);`	`// retrieve available endpoints for specified serverDiscoveryUrl.` `string serverDiscoveryUrl = applicationDescription.DiscoveryUrls[0];` `IList<EndpointDescriptionEx> endpoins = uaApplication.GetEndpoints(serverDiscoveryUrl);` *Code snippets are from DiscoveryClient sample.*
4.2	**EndpointDescription**	**EndpointDescriptionEx**
5	**Session**	**ClientSession**
5.1	*Session* instances are created using constructor. `// Create the Session object.` `Session session = new Session(serverUrl, securityMode, securityPolicy.ToString(), messageEncoding, userId, null);`	There is no public constructor for *ClientSession.* Sessions are created by *UaApplication* class with *CreateSession* methods. *Code snippet from ConnectClient sample:* `// Create the Session object.` `ClientSession session = uaApplication.CreateSession(serverUrl, securityMode, securityPolicy, messageEncoding, userId, null);` *Code snippet from AlarmsClient sample:* `// create the session object with no security and anonymous login` `ClientSession session = uaApplication.CreateSession(Constants.ServerUrl);`
5.2	**AnonymousUserIdentity** `UserIdentity userIdentity = new AnonymousUserIdentity();` **UserNameUserIdentity** `UserIdentity userIdentity = new UserNameUserIdentity("usr", "pwd");`	**AnonymousUserIdentity** class was removed. To be replaced with **UserIdentity** `UserIdentity userIdentity = new UserIdentity();` **UserNameUserIdentity** class was removed. To be replaced with **UserIdentity** `UserIdentity userIdentity = new UserIdentity("usr", "pwd");` *Code snippet from ConnectClient sample.*
5.3	**Session.HistoryContinuationPointReached**	**ClientSession** does not provide a **HistoryContinuationPointReached** event.
5.4	**RedundantSession**	Not Available
6	**Subscription**	**ClientSubscription**
6.1	**RedundantSubscription**	Not Available
7	**Browse**	
7.1	Session has an event ContinuationPointReached for controlling browse.	ClientSession ***does not*** provide the ContinuationPointReached event for controlling browse.

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
	`IList<ReferenceDescription> rootReferenceDescriptions = session.Browse(null, sender);` Browse with options: `BrowseOptions options = new BrowseOptions();` `options.MaxReferencesReturned = 3;` `IList<ReferenceDescription> rootReferenceDescriptions = session.Browse(null, options, sender);`	`IList<ReferenceDescriptionEx> rootReferenceDescriptions = m_session.Browse(null);` Browse with options: `BrowseDescriptionEx options = new BrowseDescriptionEx();` `options.MaxReferencesReturned = 3;` `IList<ReferenceDescriptionEx> rootReferenceDescriptions = m_session.Browse(null, options);` *Code snippet from BrowseClient sample.*
7.2	**BrowseOptions** `BrowseOptions options = new BrowseOptions();` `options.MaxReferencesReturned = 3;`	**BrowseDescriptionEx** `BrowseDescriptionEx options = new BrowseDescriptionEx();` `options.MaxReferencesReturned = 3;` *Code snippet from BrowseClient sample.*
7.3	Parameter *cookie* from *Browse* methods. `public virtual IList<ReferenceDescription> Browse(NodeId nodeId, object cookie);` `public virtual IList<ReferenceDescriptionEx> Browse(NodeId nodeId, BrowseDescriptionEx browseOptions, object cookie);`	Parameter *cookie* was removed from *Browse* methods. `public virtual IList<ReferenceDescriptionEx> Browse(NodeId nodeId);` `public virtual IList<ReferenceDescriptionEx> Browse(NodeId nodeId, BrowseDescriptionEx browseDescription);`
7.4	**ReferenceDescription**	**ReferenceDescriptionEx**
7.5	**BrowsePath**	**BrowsePathEx**
7.6	**BrowsePathResult**	**BrowsePathResultEx**
8	**ExpandedNodeId** to **NodeId** - wrapped classes from old SDK are used. `// transform ExpandedNodeId to NodeId` `NodeId nodeId = new NodeId(rootReferenceDescription.NodeId);`	**ExpandedNodeId** to **NodeId** - stack classes are used. `// transform ExpandedNodeId to NodeId` `NodeId nodeId = new NodeId(rootReferenceDescription.NodeId.Identifier, rootReferenceDescription.NodeId.NamespaceIndex);` *Code snippet from BrowseClient sample.*
9	**MonitoredItem** `monitoredItem = new MonitoredItem(subscription, nodeId, AttributeId.Value, null, "Sample Monitored Item");`	**ClientMonitoredItem** The order of parameters in one of the constructors changed. Also, some parameters became optional. `monitoredItem = new ClientMonitoredItem(subscription, nodeId, "Sample Monitored Item", Attributes.Value, null);` `// the code from above is similar to the next one since the default values for parameters are used`

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
		monitoredItem = new ClientMonitoredItem(subscription, nodeId, "Sample Monitored Item");
9.1	**DataValue** *Session.Read* returns *DataValue* *MonitoredItem.Read* returns *DataValue*	**DataValueEx** *ClientSession.Read* returns *DataValueEx* *ClientMonitoredItem.Read* returns *DataValueEx* *DataValueEx.TryConvertToEnumValue* method - parameter order was changed, *session* parameter was moved on the first position. When writing a value using *ClientMonitoredItem.Write* or *ClientSession.Write* the *DataValue* object used for write
9.2	Create monitored item for alarms using **EventFilter** // Configure the event filter EventFilter filter = filter = new EventFilter(); // specify the required fields of the events filter.AddSelectClause(ObjectTypes.BaseEventType, String.Empty, Attributes.NodeId); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.EventId); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.EventType); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.SourceNode); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.SourceName); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.Time); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.Message); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.Severity); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.EnabledState); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.ActiveState); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.AckedState); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.Comment); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.Retain); // filter only for condition related events (e.g in order to avoid audit events) filter.WhereClause.Push(FilterOperator.OfType, ObjectTypeIds.ConditionType); // Create the MonitoredItem used to receive	Create monitored item for alarms using **EventFilterEx** // Configure the event filter EventFilterEx filter = filter = new EventFilterEx(); // specify the required fields of the events filter.AddSelectClause(ObjectTypes.BaseEventType, String.Empty, Attributes.NodeId); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.EventId); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.EventType); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.SourceNode); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.SourceName); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.Time); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.Message); filter.AddSelectClause(ObjectTypes.BaseEventType, BrowseNames.Severity); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.EnabledState); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.ActiveState); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.AckedState); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.Comment); filter.AddSelectClause(ObjectTypeIds.AcknowledgeableConditionType, BrowseNames.Retain); // filter only for condition related events (e.g in order to avoid audit events) filter.WhereClause.Push(FilterOperator.OfType, ObjectTypeIds.ConditionType);

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
	event notifications m_alarmsMonitoredItem = new MonitoredItem(m_subscription, m_alarmsModuleNodeId, "Alarms module", filter); m_alarmsMonitoredItem.EventsReceived += AlarmsMonitoredItem_EventsReceived;	// Create the MonitoredItem used to receive event notifications m_alarmsMonitoredItem = new ClientMonitoredItem(m_subscription, m_alarmsModuleNodeId, "Alarms module", filter); m_alarmsMonitoredItem.EventsReceived += AlarmsMonitoredItem_EventsReceived;
10	**AttributeId**	**AttributeId** enumeration was removed. Use constants with matching names from **Attributes** class.
11	**DataTypeNode** **MethodNode** **ObjectNode** **ObjectTypeNode** **ReferenceTypeNode** **VariableNode** **VariableTypeNode** **ViewNode**	**DataTypeNodeEx** **MethodNodeEx** **ObjectNodeEx** **ObjectTypeNodeEx** **ReferenceTypeNodeEx** **VariableNodeEx** **VariableTypeNodeEx** **ViewNodeEx**
12	**Complex types/enum values**	
12.1	**Read complex value/enum value** Enable custom types decoding and type dictionaries reload by setting configuration value before creating the session: Application.Configuration.Client.DecodeCustomDataTypes = true; For enumeration values you need to call *TryConvertToEnumValue on DataValue* returned by session.Read. **DataValue.Value** is a *StructuredValue or EnumValue* instance if the complex value was read and it is set to something different from *null*.	**Read complex type/enum value** Enable custom types decoding and type dictionaries reload by setting configuration value before creating the *ClientSession* (by code of in XML configuration file): UaApplication.ClientToolkitConfiguration.DecodeCustomDataTypes = true; UaApplication.ClientToolkitConfiguration.DecodeDataTypeDictionaries = true; // trigger loading by calling session.FetchDataDictionaries(); // types are done loading when session.TypeDictionariesLoaded and/or DataTypeDefinitionsLoaded flags returns true For enumeration values you need to call *TryConvertToEnumValue on DataValueEx* returned by session.Read. The order of parameters for *TryConvertToEnumValue* changed. **DataValueEx.ProcessedValue** is an instance of *BaseComplexTypeValue* if a complex type value was read.
12.2	**Write complex type** //Browse path: Root\Objects\Refrigerators\Refrigerator #1\RefrigeratorStatus const string StaticValueNodeID = "ns=10;i=13"; WriteValue writeValue = new WriteValue(); writeValue.AttributeId = AttributeId.Value; writeValue.NodeId = new	**Write complex type** //Browse path: Root\Objects\Refrigerators\Refrigerator #1\RefrigeratorStatus const string StaticValueNodeID = "ns=10;i=13"; //read data type id for node StaticComplexNodeId ReadValueId readValueId = new ReadValueId(); readValueId.NodeId = new

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60

```
NodeId(StaticValueNodeID);

DataValue valueToWrite = new DataValue();

ExpandedNodeId binaryEncodingId = new
ExpandedNodeId("nsu=http://
industrial.softing.com/UA/
Refrigerator;ns=0;i=444");
ExpandedNodeId typeId = new
ExpandedNodeId("nsu=http://
industrial.softing.com/UA/
Refrigerator;ns=0;i=435");
ExpandedNodeId xmlEncodingId = new
ExpandedNodeId("nsu=http://
industrial.softing.com/UA/
Refrigerator;ns=0;i=437");
XmlQualifiedName xmlQualifiedName = new
XmlQualifiedName("RefrigeratorStatusType",
typeId.NamespaceUri);

StructuredValue complexData =
StructuredValue.CreateNew(xmlQualifiedName,
typeId, binaryEncodingId, xmlEncodingId);

complexData.AddField("CondensorMotorRunning",
true, new XmlQualifiedName("Boolean", "http://
opcfoundation.org/BinarySchema/"));
complexData.AddField("PreasureBeforePump",
(double) randomGenerator.Next(1, 110), new
XmlQualifiedName("Double", "http://
opcfoundation.org/BinarySchema/"));
complexData.AddField("PreasureAfterPump",
(double) randomGenerator.Next(1, 70), new
XmlQualifiedName("Double", "http://
opcfoundation.org/BinarySchema/"));
valueToWrite.Value = complexData;
valueToWrite.ValueRank =
ValueRanks.OneOrMoreDimensions;

writeValue.Value = valueToWrite;
try
{
    StatusCode statusCode =
session.Write(writeValue);
    Console.WriteLine("\n The NodeId:{0} was
written with the complex value {1} ",
StaticValueNodeID, writeValue.Value.ToString());
    Console.WriteLine(" Status code is {0}",
statusCode);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
```

```
NodeId(StaticValueNodeID);
readValueId.AttributeId = Attributes.DataType;

Console.WriteLine("\n Read DataType Id for
NodeId:{0}", StaticValueNodeID);

DataValueEx dataValuetypeId =
session.Read(readValueId);

//Get Default value for data type
StructuredValue defaultValue =
session.GetDefaultValueForDatatype(dataValuetypeI
d.Value as NodeId, ValueRanks.Scalar) as
StructuredValue;

WriteValue writeValue = new WriteValue();
writeValue.AttributeId = Attributes.Value;
writeValue.NodeId = new
NodeId(StaticValueNodeID);

DataValue valueToWrite = new DataValue();
valueToWrite.Value = defaultValue;
writeValue.Value = valueToWrite;
try
{
    StatusCode statusCode =
session.Write(writeValue);
    Console.WriteLine("\n The NodeId:{0} was
written with the complex value {1} ",
StaticValueNodeID, defaultValue.ToString());
    Console.WriteLine(" Status code is {0}",
statusCode);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

This sample is simplified, you do not need to know the types and their namespaces.

StructuredValue.AddField method was removed.

The *BaseComplexTypeValue* type is the base type for all complex data types. The *StructuredValue* and *EnumValue* types extend it. There are new complex data types defined: *UnionStructuredValue, OptionalFieldsStructuredValue* and *OptionSetValue.*

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
	`}`	
13	**History read**	
13.1	**Read raw** `DateTime startTime = new DateTime(2011, 1, 1, 12, 0, 0);` `DateTime endTime = new DateTime(2011, 1, 1, 12, 1, 40);` `uint numberOfValuesPerNode = 3;` `bool returnBounds = false;` `TimestampsToReturn timestampsToReturn = TimestampsToReturn.Both;` `ReadRawArgument argument = new ReadRawArgument(startTime, endTime, numberOfValuesPerNode, returnBounds, timestampsToReturn);` **Session.HistoryReadRaw** signature changed. `public virtual List<DataValue> HistoryReadRaw(NodeId nodeToReadId, ReadRawArgument readRawArgument, object cookie);`	**ReadRawArgument** class was replaced with stack class **ReadRawModifiedDetails.** `DateTime startTime = new DateTime(2011, 1, 1, 12, 0, 0);` `DateTime endTime = new DateTime(2011, 1, 1, 12, 1, 40);` `uint numberOfValuesPerNode = 3;` `bool returnBounds = false;` `TimestampsToReturn timestampsToReturn = TimestampsToReturn.Both;` `ReadRawModifiedDetails argument = new ReadRawModifiedDetails()` `{` `  StartTime = startTime,` `  EndTime = endTime,` `  NumValuesPerNode = numberOfValuesPerNode,` `  ReturnBounds = returnBounds` `};` **ClientSession.HistoryReadRaw** signature changed. The cookie parameter was removed because there is no HistoryContinuationPointReached event to match, `public virtual List<DataValueEx> HistoryReadRaw(NodeId nodeToReadId, ReadRawModifiedDetails readRawModifiedDetails, TimestampsToReturn timestampsToReturn);`
13.2	**Read at time** `List<DateTime> requiredTimes = new List<DateTime>();` `requiredTimes.Add(new DateTime(2011, 1, 1, 12, 0, 0));` `requiredTimes.Add(new DateTime(2011, 7, 1, 12, 1, 0));` `bool useSimpleBounds = true;` `TimestampsToReturn timestampsToReturn = TimestampsToReturn.Both;` `ReadAtTimeArgument argument = new ReadAtTimeArgument(requiredTimes, useSimpleBounds, timestampsToReturn);`	**ReadAtTimeArgument** class was replaced with stack class **ReadAtTimeDetails.** `DateTimeCollection requiredTimes = new DateTimeCollection();` `requiredTimes.Add(new DateTime(2011, 1, 1, 12, 0, 0));` `requiredTimes.Add(new DateTime(2011, 7, 1, 12, 1, 0));` `ReadAtTimeDetails argument = new ReadAtTimeDetails()` `{` `  ReqTimes = requiredTimes,` `  UseSimpleBounds = true` `};` `TimestampsToReturn timestampsToReturn = TimestampsToReturn.Both;`

Id.	dataFEED .NET SDK 1.43	dataFEED OPC UA .NET Standard SDK 2.60
	Session.HistoryReadAtTime signature changed. `public virtual List<DataValue> HistoryReadAtTime(NodeId nodeToReadId, ReadAtTimeArgument readAtTimeArgument, object cookie);`	**ClientSession.HistoryReadAtTime** signature changed. The cookie parameter was removed because there is no HistoryContinuationPointReached event to match, `public virtual List<DataValueEx> HistoryReadAtTime(NodeId nodeToReadId, ReadAtTimeDetails readAtTimeDetails, TimestampsToReturn timestampsToReturn);`
13.3	**Read processed** `DateTime startTime = new DateTime (2011, 1, 1, 12, 0, 0);` `DateTime endTime = new DateTime (2011, 1, 1, 12, 1, 40);` `double processingInterval = 10000;` `List<NodeId> aggregateTypes = new List<NodeId>();` `aggregateTypes.Add(new NodeId(2342)); // aggregate function average` `TimestampsToReturn timestampsToReturn = TimestampsToReturn.Both;` `ReadProcessedArgument argument = new ReadProcessedArgument(startTime, endTime, processingInterval, aggregateTypes, timestampsToReturn);` **Session.HistoryReadAtTime** signature changed. `public virtual List<DataValue> HistoryReadProcessed(NodeId nodeToReadId, ReadProcessedArgument readProcessedArgument, object cookie);`	**ReadProcessedArgument** class was replaced with stack class **ReadProcessedDetails.** `NodeIdCollection aggregateTypes = new NodeIdCollection();` `aggregateTypes.Add(ObjectIds.AggregateFunction_Average); //aggregate function average` `ReadProcessedDetails argument = new ReadProcessedDetails()` `{` `  StartTime = new DateTime(2011, 1, 1, 12, 0, 0),` `  EndTime = new DateTime(2011, 1, 1, 12, 1, 40),` `  ProcessingInterval = 10000,` `  AggregateType = aggregateTypes` `};` `TimestampsToReturn timestampsToReturn = TimestampsToReturn.Both;` **ClientSession.HistoryReadProcessed** signature changed. The cookie parameter was removed because there is no HistoryContinuationPointReached event to match, `public virtual List<DataValueEx> HistoryReadProcessed(NodeId nodeToReadId, ReadProcessedDetails readProcessedDetails, TimestampsToReturn timestampsToReturn);`
14	**Async Method calls -** *CallCompleted* event definition `public event EventHandler<MethodExecutionEventArgs> CallCompleted;` **MethodExecutionArgs** `private void Session_CallCompleted(object sender, MethodExecutionArgs e)` `{` `  ...` `}`	**Async Method calls -** *CallCompleted* event definition changed because *MethodExecutionArgs* name changed to *MethodExecutionEventArgs*. `public event EventHandler<MethodExecutionEventArgs> CallCompleted;` **MethodExecutionEventArgs -** does not have Ex suffix it is not a wrapper class. Name changed according to EventArgs classes naming rules. *CallCompleted* event handler definition will change to: `private void Session_CallCompleted(object sender, MethodExecutionEventArgs e)`

<table>
<tr><th>Id.</th><th>dataFEED .NET SDK 1.43</th><th>dataFEED OPC UA .NET Standard SDK 2.60</th></tr>
<tr><td></td><td></td><td>

```
{
    ...
}
```

</td></tr>
<tr><td>15</td><td>

Logging

The old SDK uses the following log messages from *Softing.Opc.Ua.Sdk.Trace* class.

```
public void Log(TraceLevels traceLevel,
TraceMasks traceMask, string objectId, string
message);
public void Log(TraceLevels traceLevel,
TraceMasks traceMask, string objectId, string
message, Exception exception);
public void Log(TraceLevels traceLevel,
TraceMasks traceMask, string objectId, string
message, Exception exception,
IMaskFormatProvider provider);
public void Log(TraceLevels traceLevel,
TraceMasks traceMask, string objectId, string
message, Exception exception,
IMaskFormatProvider provider, bool
bypassTraceEventHandler);
```

Logging configuration:.

```
<TraceConfiguration>
  <!-- Enable ALL -->
  <TraceMasks>0x00FF00FF</TraceMasks>
  <Log4NetConfiguration>
    <log4net>
      <appender name="RollingLogFileAppender"
type="log4net.Appender.RollingFileAppender">
        <file value="Logs\AlarmsClient.txt" />
        <appendToFile value="true" />
        <maxSizeRollBackups value="5" />
        <maximumFileSize value="10MB" />
        <rollingStyle value="Size" />
        <staticLogFileName value="true" />
        <layout
type="log4net.Layout.PatternLayout">
          <header value="[Header]&#13;&#10;" />
          <footer value="[Footer]&#13;&#10;" />
          <conversionPattern value="%date %
thread %-5level - %message%newline" />
        </layout>
      </appender>
      <root>
        <level value="WARN" />
        <appender-ref
ref="RollingLogFileAppender" />
      </root>
    </log4net>
  </Log4NetConfiguration>
</TraceConfiguration>
```

</td><td>

Logging

dataFEED OPC UA .NET Standard SDK uses the following log messages from stack class *Softing.Opc.Utils* class.

```
public static void Trace(int traceMask, string
format, bool handled, params object[] args);
public static void Trace(int traceMask, string
format, params object[] args);
public static void Trace(Exception e, string
format, bool handled, params object[] args);
public static void Trace(Exception e, string
format, params object[] args);
public static void Trace(string format, params
object[] args);
```

Logging configuration:

```
<OutputFilePath>%CommonApplicationData%\Softing
\OpcUaNetStandardToolkit\logs\SampleClient.log</
OutputFilePath>
  <DeleteOnLoad>true</DeleteOnLoad>
  <!-- Show Only Errors -->
  <!-- <TraceMasks>1</TraceMasks> -->
  <!-- Show Only Security and Errors -->
  <!-- <TraceMasks>513</TraceMasks> -->
  <!-- Show Only Security, Errors and Trace -->
  <!-- <TraceMasks>515</TraceMasks> -->
  <!-- Show Only Security, COM Calls, Errors and
Trace -->
  <!-- <TraceMasks>771</TraceMasks> -->
  <!-- Show Only Security, Service Calls, Errors
and Trace -->
  <!-- <TraceMasks>523</TraceMasks> -->
  <!-- Show Only Security,
ServiceResultExceptions, Errors and Trace -->
  <TraceMasks>519</TraceMasks>
</TraceConfiguration>
```

</td></tr>
</table>

3 Server API Migration Guide

Starting with version 2.00 of dataFEED OPC UA .NET Standard SDK we provide a simplified API for server development.

Note: Each application that uses dataFEED OPC UA .NET Standard SDK version 2.60 must add reference to NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.363.49. The nuget package contains the OPC UA stack implementation maintained by OPC Foundation under GIT repository.

The steps to migrate to using this API are listed below:

General

1. Copy your solution folder to a new location.
2. Open copy solution in Visual Studio 2017.
3. Add reference to Softing.Opc.Ua.Server.dll to your project.
4. Add reference to NuGet package OPCFoundation.NetStandard.Opc.Ua - version 1.4.363.49.
5. Refactor Server class as described in **section 1** from table below.
6. Refactor node manager classes as described in **section 2** from table below.
7. Refactor the code for starting the server as described in **section 3** from table below.

Id.	dataFEED OPC UA .NET Standard SDK 1.10	dataFEED OPC UA .NET Standard SDK 2.60
1	**Refactor Server class**	
1.1	Server class is derived from *Opc.Ua.Server.**StandardServer*** class.	Derive Server class from *Softing.Opc.Ua.Server.**UaServer*** class.
1.2	*CreateMasterNodeManager*() method override	From *CreateMasterNodeManager*() method override remove any code regarding *ShutdownDelay.* *Shutdown Delay* is handled Internaly by the OPC UA .NET Standard Stack from OPC Foundation using *ShutdownDelay* property defined in *ApplicationConfiguration*. Use that configuration property to manage your server shut down delay.
1.3	*OnServerStarted*() method override	If your implementation of *OnServerStarted*() method override does only registration of known variable and object types from current assembly and handles *SessionManager.ImpersonateUser* event this method can be removed. The registration of variable and object types from current assembly is handled by current implementation of *UaServer* class. Remove any code that does type registration and use the *RegisterTypes*() method from *UaServer* instead. For more details about registering types please read UaServer Class - **Register Types** section. The handling of *SessionManager.ImpersonateUser* event was replaced by three methods that can be

Id.	dataFEED OPC UA .NET Standard SDK 1.10	dataFEED OPC UA .NET Standard SDK 2.60
		overridden in order to provide authentication: *ValidateUserPassword(), ValidateUserCertificate() and ValidateIssuedIdentity()*. For more details please read UaServer Class - **User Authentication** section.
1.4	*OnServerStopping*() method override	If your implementation of *OnServerStopping()* method handles only the shutdown delay this method can be removed (see **section 1.2**)
1.5	*LoadServerProperties*() method override	Current implementation of *LoadServerProperties*() method from UaServer creates a new ServerProperties instance using this code: `protected override ServerProperties LoadServerProperties()` `{` `  ServerProperties properties = new ServerProperties();` `  properties.ManufacturerName = ManufacturerName;` `  properties.ProductName = Configuration.ApplicationName;` `  properties.ProductUri = Configuration.ProductUri;` `  properties.SoftwareVersion = Utils.GetAssemblySoftwareVersion();` `  properties.BuildNumber = Utils.GetAssemblyBuildNumber();` `  properties.BuildDate = Utils.GetAssemblyTimestamp();` `  return properties;` `}` Override *LoadServerProperties*() method only if needed, otherwise set the desired properties in configuration.
2	**Refactor node manager classes**	
2.1	Node manager classes are derived from *Opc.Ua.Server.**CustomNodeManager2***.	Derive node manager classes from *Softing.Opc.Ua.Server.**NodeManager*** class.
2.2	*CreateAddressSpace()* method override	Continue to use *CreateAddressSpace()* method override to create the static address space. Refactor the code to use the helper methods from dataFEED OPC UA .NET Standard SDK's *NodeManager* class for creating object, variable and method nodes as described in Create Static Address Space.
2.3	*GetManagerHandle()* and *ValidateNode()* methods override	*GetManagerHandle()* and *ValidateNode()* methods implementation from dataFEED OPC UA .NET Standard

<table>
<tr><th>Id.</th><th>dataFEED OPC UA .NET Standard SDK 1.10</th><th>dataFEED OPC UA .NET Standard SDK 2.60</th></tr>
<tr><td></td><td></td><td>SDK's NodeManager class provides desired functionality for all node managers. There is no need to override these methods and they can be removed from your node manager implementation.</td></tr>
<tr><td>2.4</td><td>New() method override</td><td>Default implementation of New() method provides unique NodeId instances with numeric identifier.
Override New() method only if different NodeId creation logic is needed. For more details please read Node Managers - New() method override section.</td></tr>
<tr><td>2.5</td><td>Create object, variable and method nodes</td><td>Refactor the code to use the helper methods from dataFEED OPC UA .NET Standard SDK's NodeManager class for creating object, variable and method nodes as described in Create Static Address Space.</td></tr>
<tr><td>3</td><td colspan="2">Start the Server</td></tr>
<tr><td>3.1</td><td>Previous version of dataFEED OPC UA .NET Standard SDK used an instance of ApplicationInstance class from OPC UA .NET Standard Stack from OPC Foundation to start a server:
<pre>
ApplicationInstance application = new
ApplicationInstance();
application.ApplicationType =
ApplicationType.Server;
string configurationFile =
"SampleServer.Config.xml";

try
{
    // Load the application configuration
    await
application.LoadApplicationConfiguration(configu
rationFile, false);

    // Check the application certificate
    await
application.CheckApplicationInstanceCertificate(
false, 0);
    application.ApplicationConfiguration.Certifi
cateValidator.CertificateValidation +=
CertificateValidator_CertificateValidation;
    // Start the server
    await application.Start(new SampleServer());
    do
    {
        ConsoleKeyInfo key = Console.ReadKey();
        if (key.KeyChar == 'q' || key.KeyChar
== 'x')
        {
</pre></td><td>Starting from version 2.00 of the dataFEED OPC UA .NET Standard SDK, the new UaServer class implementation provides three methods for starting the server:
<pre>
string configurationFile =
"SampleServer.Config.xml";
SampleServer sampleServer = new SampleServer();
try
{
    // Start the server
    await sampleServer.Start(configurationFile);

    do
    {
        ConsoleKeyInfo key = Console.ReadKey();
        if (key.KeyChar == 'q' || key.KeyChar ==
'x')
        {
            break;
        }
    }
    while (true);
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
    Console.ReadKey();
    Environment.Exit(-1);
}
</pre>
For more details about registering types please read UaServer Class - Start the Server section.</td></tr>
</table>

Id.	dataFEED OPC UA .NET Standard SDK 1.10	dataFEED OPC UA .NET Standard SDK 2.60
	`` break; } } while (true); } catch (Exception e) { Console.WriteLine(e.ToString()); Console.ReadKey(); Environment.Exit(-1); } ```	

4 Copyrights

The dataFEED OPC UA .NET Standard SDK uses the following 3rd party libraries:

OPC Foundation:

- **RCL -** .NET Standard Stack is used under the RCL license. The RCL license agreement can be found at https://opcfoundation.org/license/rcl.html
  Softing is sub licensing the .NET Standard Stack under the RCL license to its dataFEED OPC UA .NET Standard SDK customers. This allows the dataFEED OPC UA .NET Standard SDK customers to distribute the .NET Standard.
- **MIT -** The MIT license agreement can be found at https://opcfoundation.org/license/mit.html

- **OpenSSL**
  The OpenSSL license agreement can be found at https://www.openssl.org/source/license-openssl-ssleay.txt
- **Bouncy Castle**
  The Bouncy Castle license agreement can be found at https://www.bouncycastle.org/license.html