Technical Comparison: *dataFEED OPC UA C++ SDK* V5.56 vs. *open62541* V0.3.0

Preliminary Remarks

This document provides a comparison between the Softing ***dataFEED OPC UA C++ SDK*** V5.56 (Linux version available at https://data-intelligence.softing.com/products/datafeed-opc-sdks/datafeed-opc-ua-c-server-client-sdk-for-linux/) and the free open source OPC UA implementation ***open62541*** V0.3.0 (available at https://github.com/open62541/open62541).

The comparison addresses OPC UA Clients as well as OPC UA Servers developed using these toolkits.

OPC UA Client Functionality

The table below compares the OPC UA Client functionality as provided by ***dataFEED OPC UA C++ SDK*** and ***open62541***.

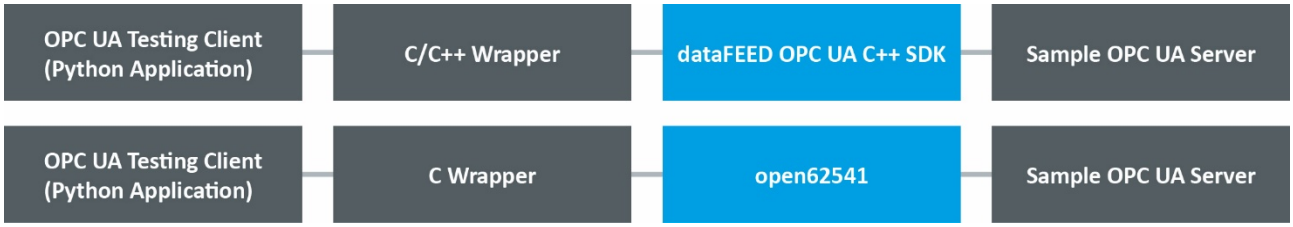
Client Functionality	***dataFEED OPC UA C++ SDK* Support**	***open62541* Support**
Subscription	Good	Poor
Security Profile: None / None	Good	Good
Security Profile: Sign / Basic256Sha256	Good	Good
Security Profile: Sign&Encrypt / Basic256Sha256	Good	Good
Anonymous authentication	Good	Good
Username and password authentication	Good	Good
User certificate authentication	Good	Poor

OPC UA Client Testing Environment

The OPC UA Clients based on ***dataFEED OPC UA C++ SDK*** as well as ***open62541*** both are implemented as a Phyton application using a C wrapper for accessing the toolkits' C++ API.

The OPC UA Client tests have been performed together with a sample OPC UA Server developed using the Softing ***dataFEED OPC UA C++ SDK***. It has been configured as follows:

- 1,000,000 nodes of data type Int16
- Address space of 10,000 nodes, structured as one object with 100 folders
  The address space has been defined by an OPC UA Modeling editor and stored in an XML file. This XML file then has been loaded into the sample OPC UA Server.



Softing Industrial Automation GmbH
Richard-Reitzner-Allee 6
85540 Haar / Germany
https://data-intelligence.softing.com
+ 49 89 4 56 56-113
+ 49 89 4 56 56-606
info.idi@softing.com

OPC UA Client Performance Tests

During the OPC UA Client performance test the time required to perform the OPC UA services Browse, Read, Write and Subscribe/Monitor is measured.

1 Browse Test

During the browse test the OPC UA Server nodes of the address space object are browsed recursively. The browse request is sent as a single request. In addition, the variable nodes are browsed.

Number of Browsed Nodes	*dataFEED OPC UA C++ SDK* Client	*open62541* Client
100,000	1:01 min	0:56 min
1,000,000	8:58 min	8:52 min

2 Read Test

The read test measures the time to read a total of 1,000,000 nodes, split up into requests to read a specific number of nodes.

Number of Nodes Read per Request	*dataFEED OPC UA C++ SDK* Client	*open62541* Client
1,000 / request	16 s	7 s
5,000 / request	16 s	Client doesn't support read services for requests to read more than 3,630 nodes. In this case it sends improperly formatted telegrams to the server.
10,000 / request	17 s	

3 Write Test

The write test measures the time to write a total of 1,000,000 nodes, split up into requests to write a specific number of nodes.

Number of Nodes Written per Request	*dataFEED OPC UA C++ SDK* Client	*open62541* Client
1,000 / request	18 s	7 s
5,000 / request	19 s	5 s
10,000 / request	18 s	5 s
50,000 / request	16 s	6 s
100,000 / request	17 s	Crash client doesn't support to write this number of nodes within one request

4 Subscribe/Monitor Test

The subscribe/monitor test has been aborted due to improperly formatted publish requests sent by the ***open62541*** client. In addition, the ***open62541*** client stops to send publish requests after few seconds and thus doesn't receive any data change events.

Softing Industrial Automation GmbH
Richard-Reitzner-Allee 6
85540 Haar / Germany
https://data-intelligence.softing.com
+ 49 89 4 56 56-113
+ 49 89 4 56 56-606
info.idi@softing.com

OPC UA Server Functionality

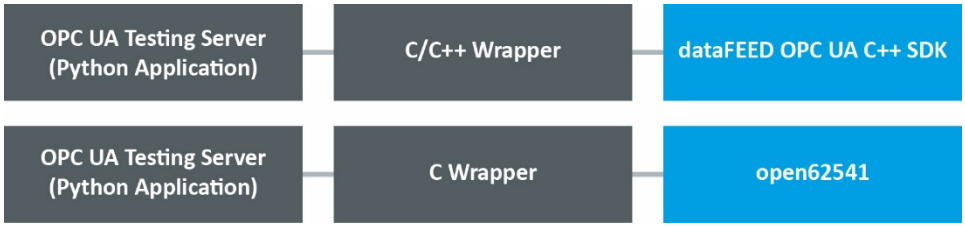
The table below compares the OPC UA Server functionality as provided by ***dataFEED OPC UA C++ SDK*** and ***open62541***.

Server Functionality	*dataFEED OPC UA C++ SDK* Support	*open62541* Support
Creating address space with 1,000,000 nodes	Good	Poor
Security Profile: Sign / Basic256Sha256	Good	Good
Security Profile: Sign&Encrypt / Basic256Sha256	Good	Poor
Anonymous authentication	Good	Good
Username and password authentication	Good	Good
User certificate authentication	Good	Poor
Trusted List	Good	Poor
Revocation List	Good	Poor

OPC UA Server Testing Environment

The OPC UA Servers based on ***dataFEED OPC UA C++ SDK*** as well as ***open62541*** both are implemented as a Phyton application using a C wrapper for accessing the toolkits' C++ respectively C API.

The OPC UA Server tests have been performed based on an address space of 1,000,000 nodes, stored in an XML file.



OPC UA Server Performance Tests

- ***dataFEED OPC UA C++ SDK* Server**

  The ***dataFEED OPC UA C++ SDK*** server starts after a few seconds with an address space of 1,000,000 nodes.

- ***open62541* Server**

  The ***open62541*** server failed to start with an address space of 1,000,000 nodes.
  The following approaches have been tried:

  a) Create of address space C file using OPC UA Modeling editor
  The C file can be linked to the implemented OPC UA Server, afterwards the converter crashed due to too less memory.
  (Reference: open62541/tools/nodeset_compiler/nodeset_compiler.py)

  b) Write address space C file directly using Phyton code and compile it together with OPC UA Server code
  The compiler crashed due to too less memory. (The testing PC was equipped with 32 GB RAM.)

  c) Start implemented OPC UA Server without address space and add address space dynamically using Phyton application
  After about 30 minutes the test was stopped as the OPC UA Server was still busy to create the address space and thus the OPC UA Server still could not be used.

Conceptual Issues

This section gives an overview about some conceptual, architectural and implementation differences between ***dataFEED OPC UA C++ SDK*** and ***open62541***. The main purpose of this architectural comparison is to identify possible reasons for performance differences and for recognized functionality restrictions.

1 **Supported Features** (not complete)

Supported Features	***dataFEED OPC UA C++ SDK***	***open62541***
Historical access	Good Fully supported but requires custom implementation and detailed OPC UA know-how	Poor Partially prepared but not yet supported
Views	Poor Views are not supported for browsing	Poor Views are not supported for browsing
Toolkit extensions	Neutral Core functionality not extensible by applications Cyclic callbacks are not necessary, as toolkit is designed for multi-threading.	Good Plug-in (well-known structure with function pointers for access handling) for easy replacement by application available; custom implementation may initialize function pointers in a different way Support of registration of cyclically invoked callbacks, helping to implement plug-ins (see NodeStore comparison)
Default application configuration	Good Application has to provide full configuration for important parts This approach requires more effort but shows necessary actions	Poor “Too simple” convenience functions for application initialization (just fitting to sample application, not to productive OPC UA Clients and Servers) It is likely to miss important configuration settings (for instance Application URI urn:unconfigured:application: Application URI has to be unique within network)

Supported Features (continued)

Supported Features	*dataFEED OPC UA C++ SDK*	*open62541*
User Access Control	Good Built-in concept to manage access rights for different user groups.	Poor No built-in support to handle access rights for different user groups (Writing plug-in to grant access to all nodes for certain users would be simple, granting individual access to individual nodes requires to design and implement a concept for how to store and access the access rights per node or NodeId. Customers currently have to implement their own plug-in including concept how to configure user rights for individual nodes. It will require some time to implement this feature.)

Softing Industrial Automation GmbH
Richard-Reitzner-Allee 6
85540 Haar / Germany
https://data-intelligence.softing.com
+ 49 89 4 56 56-113
+ 49 89 4 56 56-606
info.idi@softing.com

2 Overall Architecture

dataFEED OPC UA C++ SDK	*open62541*
Implementation Details	
Several layers split into different libraries: • C++ API for application access • CAPI + Core to separate internal functionality • Stack handling communication • OpenSSL Only C++ API, stack, OpenSSL accessible from applications (direct stack access doesn't make sense), Core implementation is completely hidden (except for source code delivery).	Complete implementation within one code base (one project) Few logical layers, depending on purpose for data access: • Convenience API functions • "Normal" API functions for full functionality • Internally used functions • Plug-in implementations (where available) • Wolf SSL Complete implementation accessible, visibility only limited by used header files
Some application wide information provided by singletons	Every OPC UA Client or Server is instance of structure, is required as input parameter for every API access function
	Server toolkit accesses its own address space in same manner as client requests by using special "admin" session
Advantages / Disadvantages	
Disadvantage Passing data from application to stack or vice versa usually requires mapping same data to different data structures several times • Copying decreases performance and memory footprint • Remapping different data structures increases code size	Advantage Encoder / decoder can use data structures as provided / filled by API Good for performance and memory footprint
Neutral Application can override API implementation, but not internal code	Advantage Default plug-ins can be replaced easily (e.g. retrieve the server's node data from aggregated server or data base)
Neutral Only one OPC UA application within the same process	Advantage Allows multiple OPC UA applications within same process

3 NodeStore

dataFEED OPC UA C++ SDK	*open62541*
Implementation Details	
Node map is implemented within OTServerAddressSpaceRoot in Core	NodeStore is implemented in default plug-in
Access is maintained by methods of OTServerAddressSpaceRoot	NodeStore plug-in interface is structure with function pointers for • Create • Delete • Get • Release • Copy • Insert • Replace • Iterate
Nodes are stored in std::map, storing entries in red-black-tree	Nodes are stored in 32-bit hash map implementation (including hash collision handling, memory reallocation on big size changes, etc.)
Node entries are stored by reference counting pointer ensuring life time of used memory Reference count is used for any kind of pointers and for references	Node entries contain “refcount” (i.e. currently acquired via get()) and “deleted” flag Nodes can be marked for deletion and releasing last refcount invokes deletion
Synchronized access	Synchronized access (when compiled for multi-threading)
Entire Core implementation is inaccessible from API.	Node store is designed to be used only internally, application access to nodes only via top-level API
Advantages / Disadvantages	
Neutral Search algorithm of std::map is still very fast, but for huge address spaces it is slower than hash map	Advantage Hash map allows searching nodes via NodeIds in the fastest possible way
Disadvantage Node map implemented in Core and cannot be replaced	Advantage Node plug-in can be replaced easily for accessing node data in a different manner

4 Variants

dataFEED OPC UA C++ SDK	*open62541*
Implementation Details	
ValueStruct stores content in union (either raw or as pointer).	Variant stores its data in VOID pointer
Data type is identified by a number (enumeration) This number is used to map data types to correct functions (e.g. for encode / decode).	Data type is identified by pointer to global stored structure, identifying and describing used data type Global data type describing structures are stored in array with well-known indexes Data type describing structure points to other data type describing nested types
	Contains “StorageType” field, identifying whether content data is owned or just referenced
Advantages / Disadvantages	
Disadvantage Calling a getter function through several layers often requires to copy entire value content on every layer This can be prevented by returning Variants via output parameter instead of using return value of a function	Advantage Copying a Variant does not need to copy the content unless it is explicitly required (e.g. for making a safe copy before the source content is modified) Local variables of certain data type can be stored into a Variant without copying the content (e.g. for calling a setter function)
Neutral Encoder must map identifier to various well-known encoder functions • No penalty in performance for well-known structures • Custom structure data type has to be searched (but binary search is fast enough) Adding new data types requires to adapt all places mapping identifiers to functions	Advantage Encoder doesn’t have to search for matching encoding functions as these can be accessed via data type describing structure No difference between handling of well-known structures and custom structures
Neutral Well-known data types require encoding and decoding function calling simple data type encoder / decoder resulting in increased code size	Advantage Only simple (final) data types require encoding and decoding function, all other simply refer to encoder or nested data type or continue on the next nesting level
Neutral Comparing data types only requires comparison of identifiers	Neutral Comparing data types requires comparison of pointers to data type structures within global array Data type structure can be selected via the well-known array index
Advantage Data type identifier is same as numeric identifier of data type NodeId.	Disadvantage Index in data type array is different from NodeId identifier of data type, potentially causing confusion

5 VariableNode

dataFEED OPC UA C++ SDK	*open62541*
Implementation Details	
Every variable stores DataValue for current value	VariableNode has option to either store a cache value or a structure with function pointers for a getValue() and setValue() handler This is implemented as union
Advantages / Disadvantages	
Neutral Cache value can be used to store fixed default values or to be filled with updated data	Neutral Cache value can be used to store fixed default values or to be filled with updated data
Neutral Application can override virtual callback methods to access current data from data source	Neutral Application can register callbacks to access current data from data source
	Advantage Cache memory is saved when values are provided via callbacks
Advantage Applications can implement a mixture of using cache value and requesting data from data source, being useful when dealing with MaxAge parameter of ReadService	Disadvantage Implementing a mixture of cache and custom handling requires to maintain callbacks cache Callbacks don't provide read parameter for MaxAge Reading data requires access to data source in any case

6 NodeReferences

dataFEED OPC UA C++ SDK	*open62541*
Implementation Details	
Every node stores multimap of references, the key being reference type plus BrowseDirection. Every reference stores information about reference type, direction and pointers to source and target node.	Every node stores an array of a UA_NodeReferenceKind, identifying one reference type and the direction Every UA_NodeReferenceKind has list of target NodeIds.
Advantages / Disadvantages	
	Advantage Grouping references by ReferenceKind slightly reduces required memory when having several references of the same kind
Neutral Search speed for searching references is affected by total number of references on node (binary search) and by number of references to target node (linear search) Search speed is sufficient for most use cases	Neutral Search speed for searching references is affected by amount of reference kinds (linear search) and by number of referenced nodes of selected kind (linear search) Linear search is sufficient for most use cases

7 Dynamic Creation of Nodes

dataFEED OPC UA C++ SDK	***open62541***
Implementation Details	
Providing functions to generate instances according to used type node Created nodes get generated GUID NodeIds	Providing functions to generate instances according to used type node Created nodes get generated numeric NodeIds
Manufacturer provides virtual methods for dynamic creation of all necessary API objects Custom creator can override any created methods and needs to be registered at toolkit initialization	Every VariableType or ObjectType node can be configured with callbacks to allow a custom implementation during creation of instance nodes Note: Callbacks are invoked on direct type and every parent type
Advantages / Disadvantages	
Neutral Only provides fixed number of callbacks and own API class for one node type For instance, creating multiple variable instances requires selection of correct type within creator	Advantage Support of custom created implementation for every VariableType of ObjectType
Disadvantage GUID NodeIds are generated differently on every OPC UA Server restart Applications should reassign NodeIds after node creation	Neutral Generated numeric NodeIds produce same numbers on every OPC UA Server restart

Additional Implementation Issues

The table below compares additional functionality as provided by ***dataFEED OPC UA C++ SDK*** and ***open62541***.

Functionality	***dataFEED OPC UA C++ SDK* Support**	***open62541* Support**
NodeSet compiler	Neutral	Good
CTT compliance (Testlab)	Good	Poor
Documentation to setup application with less functions	Poor	Good
Enumeration datatypes	Good	Neutral

Softing Industrial Automation GmbH
Richard-Reitzner-Allee 6
85540 Haar / Germany
https://data-intelligence.softing.com
+ 49 89 4 56 56-113
+ 49 89 4 56 56-606
info.idi@softing.com

open62541 Implementation Observations

- The ***open62541*** implementation of datatypes isn't consistent with the OPC UA standard.
  For instance, ***open62541*** implements the data type BOOL as 0 while the OPC UA implementation of this data type is 1.
- ***open62541*** provides an easy implementation of small OPC UA applications while the implementation of more functionality requires an in-depth research of the toolkit source code.
  For instance, the functionality to read the value of one single variable is easy to implement. However, the functionality to read the value of multiple variables or to read other attributes but their value requires the implementation of the concept to create request and response "objects".
- The source code of the OPC UA application created by ***open62541*** is smaller than the source code of the application created by ***dataFEED OPC UA C++ SDK***.
- ***open62541*** does not support the inheritance of variable node classes or object node classes while this functionality is supported by ***dataFEED OPC UA C++ SDK***.

Softing Industrial Automation GmbH
Richard-Reitzner-Allee 6
85540 Haar / Germany
https://data-intelligence.softing.com
+ 49 89 4 56 56-113
+ 49 89 4 56 56-606
info.idi@softing.com