KVD

Persistent custom nodes for the ctrlX Data Layer.

The app provides population of

- *explicit* (configured) nodes of any kind by configuration.
- *implicit* (dynamic) **Variable** nodes, created by any Data Layer client programmatically (*create-on-write*).

A reliable and a high performance data storage is included. The persistence is based on a key/value database (*KVD*).

Quick Start

- Install the KVD app, open ctrlX UI inside your preferred browser.
- Navigate to *Settings->Data Layer*.
- Discover the samples nodes */samples/kvd/**.

Usage

Configure *explicit* nodes or root nodes for *implicit* (dynamic) child nodes (.../*, .../**), which should be populated to the Data Layer's address space. This can be done by editing configuration file *nodes.json*.

Interaction with any Data Layer client

Read a node's value

Read a node by calling the corresponding **read** method. The argument has to be the address of the node.

Write a node's value

Create the node of *nodeClass* **Variable** by calling the corresponding **write** method. The argument has to be a Variant holding a value of any data type.

create-on-write

Create a none existing *implicit* node as a child node of an *implicit* root node, by initially writing a value to it. This can be done by just writing any value, which creates a *implicit* node of *nodeClass* **Variable** with given data type.

The initial data type of the node will be set and persistent. Subsequent writes to the node's value will fail, if the data type of the written data type does not match exactly.

Create a node

Creating a node by calling the corresponding **create** method is **NOT SUPPORTED**. Please use *create-on-write*.

Delete a node

Delete a node by calling the corresponding **remove** method to remove the node. The argument has to be the address of the node.

Only supported for *implicit* nodes.

Interaction with Node-RED

Please see README-NODE-RED.md

Configuration Editors

The KVD app can be configured with the **App data editor** or via **WebDAV**.

The KVD app configuration files can be found there:

```
Settings -> Manage app data -> Key Value Database configuration
```

WARNING:

PLEASE DO NOT EDIT, DELETE OR RENAME ANY FILE INSIDE THE FOLDER WHERE THE UNDERLYING DB IS LOCATED. THIS IS FOLDER *db* BY DEFAULT.

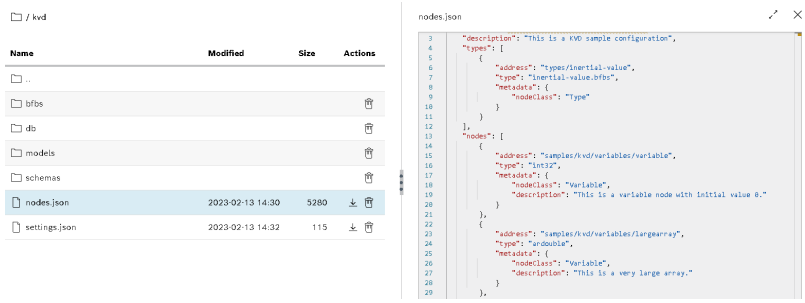
Reset

WARNING:

YOU CAN RESET THE KVD PERSISTANCE BY JUST REMOVING THE DB FOLDER. THE APP IS RESTARTED AND ALL DATA WILL LOSS AND RESETTET TO DEFAULT!

Configuration via App data editor

You can edit *settings.json*, *nodes.json* and any other there the setup the KVD app. Changes will be applied immediately.

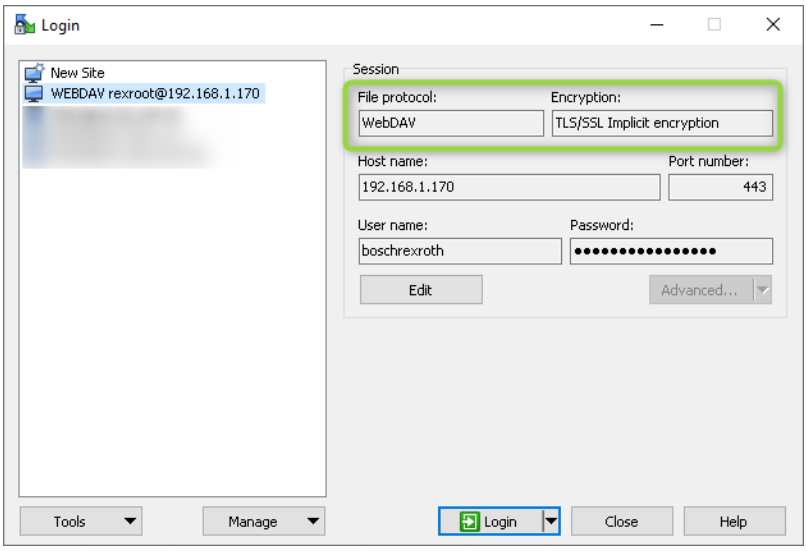


Configuration via WebDAV

Please use any **WebDAV** client to configure the KVD app.

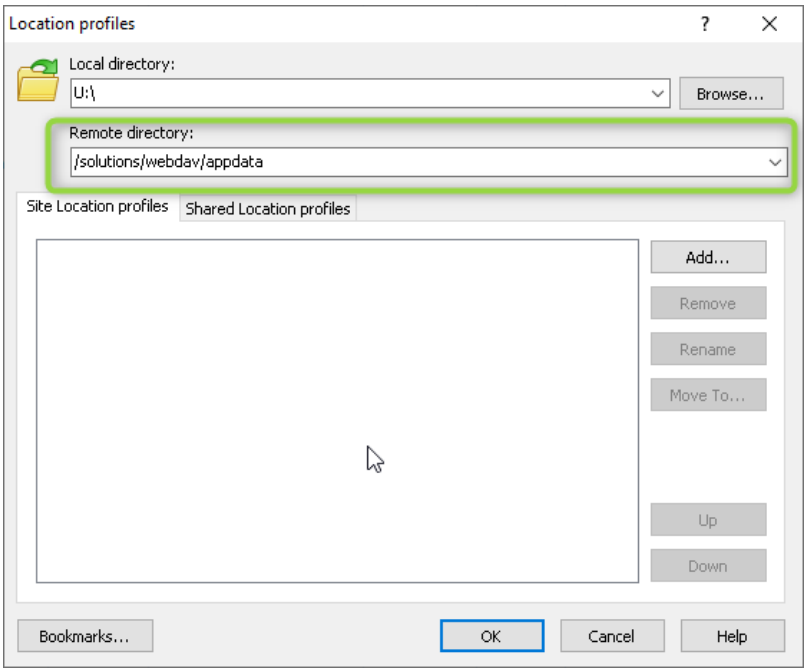
We recommend to use *WinSCP*.

Create a new site with protocol *WebDAV*, choose TLS encryption, enter your ip and credentials, finally login:

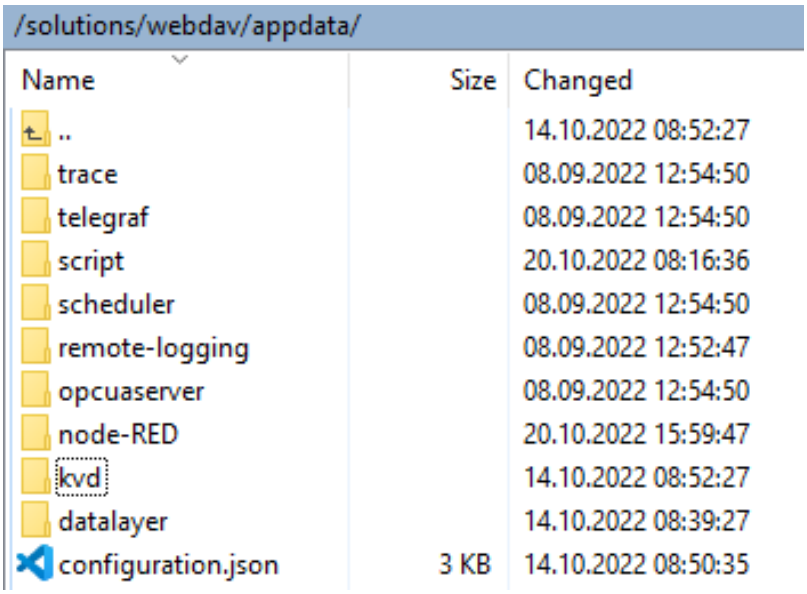


On the remote side on the left, click *Open Directory* <CTRL + o> and edit the *Remote directory* to:

```
/solutions/webdav/appdata
```

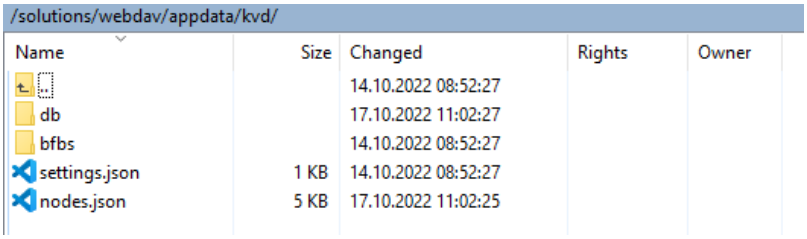


Now you should see the ctrlX appdata storage:



Navigate to folder *kvd* for configuration.

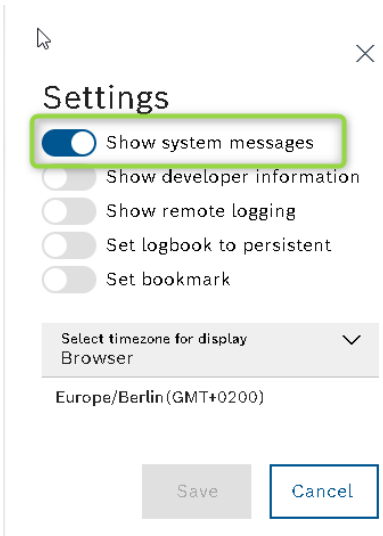
You can edit *settings.json*, *nodes.json* and any other there the setup the KVD app. Changes will be applied immediately.



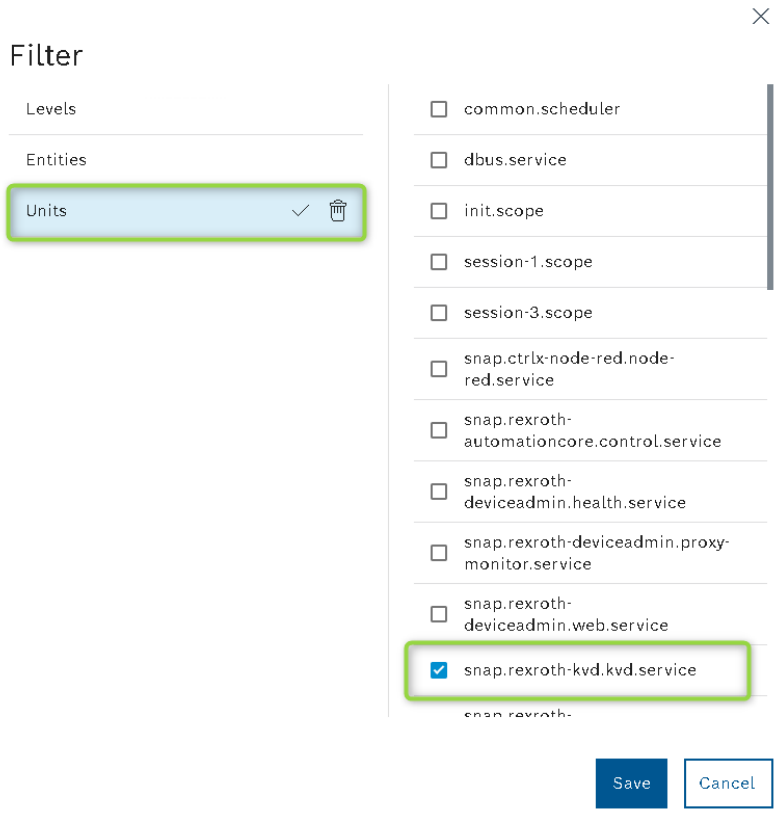
Diagnostics

On any configuration error, the KVD app doesn't startup and you won't see any nodes in the ctrlX Data Layer tree. If this is the case please check the KVD app errors in Diagnostics for error details (e.g. schema validation failed, etc.).

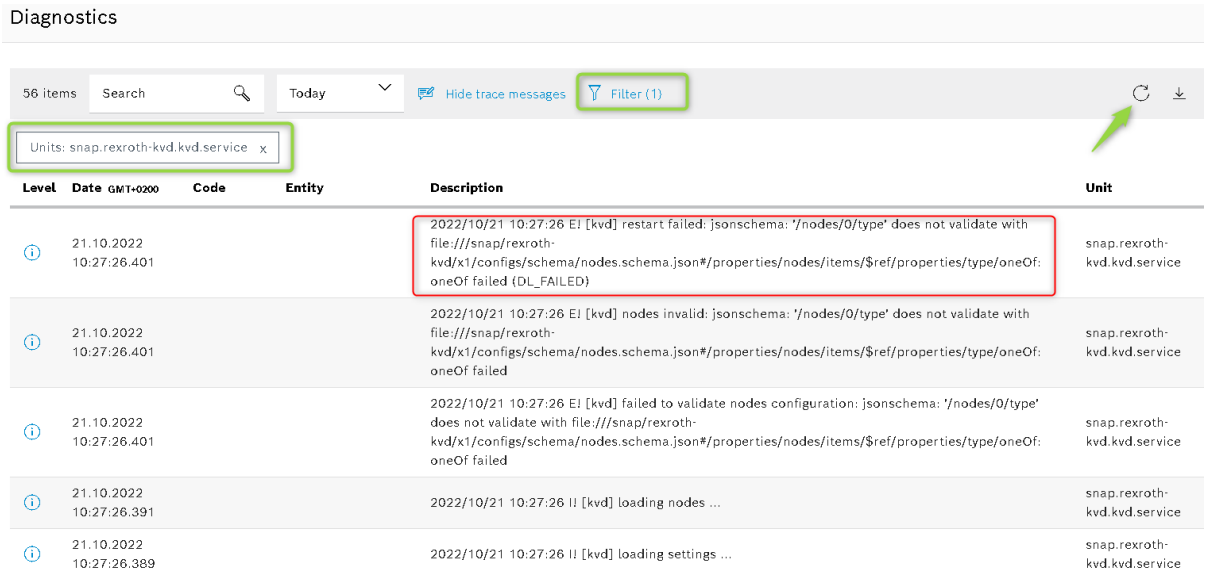
Open the ctrlX UI, navigate to *Diagnostics->Logbook*, click on the *settings* button on the right and enable option *Show system messages*.



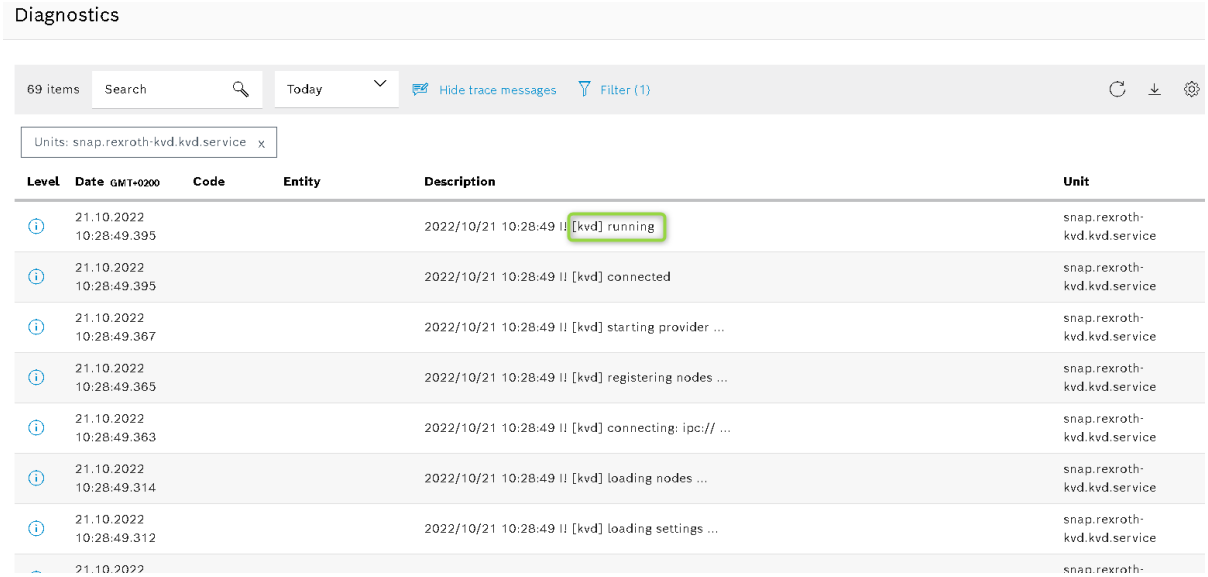
Click on *Filter* and enable Unit *snap.rexroth-kvd.kvd.service*:



If any error occured, refresh the log to see detailed error informations:



If the KVD app is starting up normally, it should look like:



General configuration

Edit the file *settings.json*, to change the general behaviour of the app.

The *settings.json* file is located in ctrlX appdata storage directory *KVD* on your ctrlX.

db

Sets the path of the underlying database. Defaults to *kvd/db*.

IMPORTANT:

PLEASE USE REMOVABLE MEDIA FOR DB STORAGE IF POSSIBLE

We recommend to use any mounted removable media (e.g. SD card) for db storage location, to keep the life time of the internal system storage.

Example: removable media

```
{
    "db": "/media/mmcblk1p1/db",
    ...
}
```

readonly

Sets the configured node values to be *readonly*, which means write operations are forbidden and will fail. Defaults to *false*.

verbose

Enables verbose trace. Defaults to *false*.

Nodes configuration

The node configuration file *nodes.json* specifies a set of nodes to be registered and persisted to the Data Layer.

The *nodes.json* file is located in ctrlX appdata storage directory *KVD* on your ctrlX.

The configuration supports *explicit* and *implicit* (dynamic) root nodes.

description

The description of the configuration.

types

The list of types (*nodeClass* **Type**)to be registered.

nodes

The list of nodes to be populated. The app supports hierarchical structuring nodes (*nodeClass* **Folder**, **Collection**, **Resource**) and variable nodes (*nodeClass* **Variable**).

address [required]

The address of the node. This can be either an absolute path (leaf), a single (**/***) or double wild carded(**/****) ending address used to specify a *implicit* node.

Volatile (memory-only) nodes

Configure a node's value to be volatile (memory-only) and not persistent by adding the extension key *autoSave* to the node's metadata, with it's value set to string *"false"*.

TIP: You can also set *volatile* to value *true* as a shortcut.

Example: Volatile nodes

```
nodes: [
    {
        "address": "mycompany/europe/myplants/plant1/volatile_temperature",
        "type": "int32",
        "metadata": {
            "nodeClass": "Variable",
            "extensions": [
                {
                    "key": "autoSave",
                    "value": "false"
                }
            ]
        }
    },
    ...
    {
        "address": "mycompany/europe/myplants/plant2/volatile_temperature",
```

```
        "type": "int32",
        "volatile": true,
    },
    ...
]
```

The volatile feature is only supported for *explicit* nodes.

Read-only nodes

Configure a node's value to be read-only (e.g. any not writable constant value) by adding the operations *write* to the node's metadata, with it's value set to *false*.

TIP: You can also set *mode* to string "*read-only*" as a shortcut.

Examples: read-only nodes

```
nodes: [
    {
        "address": "mycompany/europe/myplants/plant1/myconstant1,
        "type": "int32",
        "value": 42,
        "metadata": {
            "nodeClass": "Variable",
            "operations": {
                "write": false
            },
            "description": "This is a read-only variable node, it's value can't be
written."
        }
    },
    ...
    {
        "address": "mycompany/europe/myplants/plant1/myconstant2,
        "type": "int32",
        "value": 42,
        "mode": "read-only",
        "metadata": {
            "nodeClass": "Variable",
            "description": "This is a read-only variable node, it's value can't be
written."
        }
    }
    ...
]
```

Explicit nodes

A *explicit* node has an absolute address. A *explicit* node can't be deleted (independendly from allowed operations by the *nodeClass)*.

Example: explicit node

```
nodes: [
    {
        "address": "mycompany/europe/myplants/plant1/temperature",
        "type": "int32",
        "metadata": {
            "nodeClass": "Variable",
        }
    },
    ...
]
```

Implicit nodes

A *implicit* node can be created (*create-on-write*) and removed. A *implicit* root node defines the entry of an *implicit* branch, allowing creation and deletion of child nodes only in next level (**/***) or all levels (**/****).

Examples: Implicit root node

This *implicit* root node allows creation and deletion of child nodes only on next hierarchical level:

```
{
    "address": "mycompany/europe/myplants/plant1/*",
},
```

This *implicit* root node allows creation and deletion of child nodes of the full underlying branch (all hierarchical levels):

```
{
    "address": "mycompany/europe/myplants/plant1/**",
}
```

value

The initially preset **value** of the node. Defaults to any *falsy* value (0, false, "")

Supported values types:

```
string,number,integer,boolean,array,object
```

Examples: value

A **Variable** node of numeric value kind:

```
{
    ...
    "value": 42,
}
```

A **Variable** node of string value kind:

```
{
    ...
    "value": "42",
}
```

metadata

The metadata of the node which tells the clients something about the node's identity, allowed operations on it and more.

User access restrictions:

To configure a full *read-only* node branch or *restrict access*, please use the Data Layer user access restriction.

nodeClass

Sets the class of the node. Defaults to **Variable**.

Allowed operations like **read, write, create, delete, browse** can be executed by any client, depending on the *nodeClass* of the node.

TIP: You can also set *nodeClass* to node instead using the metadata as a shortcut.

Supported node classes:

```
Variable,Resource,Collection,Folder,Type
```

The following table shows the *allowed operations* by *nodeClass*:

Operation	Variable	Resource	Collection	Folder	Type
read	X	X	X		X
write	X	X			
create		X	X		

Operation	Variable	Resource	Collection	Folder	Type
delete		X			
browse	X	X	X	X	X

unit

The unit of the node. Defaults to an *empty* string.

TIP: You can also set *unit* to node instead using the metadata as a shortcut.

displayName

The display name of the node. Defaults to *address*.

TIP: You can also set *displayName* to node instead using the metadata as a shortcut.

displayFormat

The display format of the node. Defaults to *Auto*.

TIP: You can also set *displayFormat* to node instead using the metadata as a shortcut.

Supported display formats:

```
Auto,Bin,Oct,Dec,Hex
```

description

The description of the node. Defaults to an *empty* string.

TIP: You can also set *description* to node instead using the metadata as a shortcut.

descriptionUrl

The description url of the node. Defaults to an *empty* string.

TIP: You can also set *descriptionUrl* to node instead using the metadata as a shortcut.

Example: Minimal metadata

Numeric **Variable** node (int32):

```
nodes: [
    {
        "address": "mycompany/europe/myplants/plant1/temperature1",
        "type": "int32",
        "metadata": {
            "nodeClass": "Variable"
```

```
        }
    },
    ...
    {
        "address": "mycompany/europe/myplants/plant1/temperature2",
        "type": "int32",
        "nodeClass": "Variable"
    },
    ...
]
```

Examples: All metadata

Numeric **Variable** node (int32):

```
nodes: [
    {
        "address": "mycompany/europe/myplants/plant1/temperature",
        "type": "int32",
        "value": 42,
        "metadata": {
            "nodeClass": "Variable",
            "displayName": "This is a velocity variable node for binary experts.",
            "displayFormat": "Bin",
            "unit": "m/s",
            "description": "This is a variable node with all metadata.",
            "descriptionUrl": "https://www.boschrexroth.com/"
        }
    },
    ...
]
```

Flatbuffers **Variable** node of a **built-in** type

```
nodes: [
    {
        "address": "mycompany/europe/myplants/plant1/my-server-settings",
        "type": "types/datalayer/server-settings",
        "metadata": {
            "nodeClass": "Variable",
            "description": "This node is of built-in flatbuffers type 'server-
settings'.",
        }
    },
    ...
]
```

Flatbuffers **Variable** node of a **custom** type:

```
nodes: [
    {
        "address": "mycompany/europe/myplants/plant1/my-inertial-value",
        "type": "types/inertial-value",
        "metadata": {
            "nodeClass": "Variable",
            "description": "This node is of the custom flatbuffers type
'types/inertial-value."
    },
    ...
]
```

type

The data type of the node's value. Defaults to *unknown*.

Scalars

type	description
unknown	unknown datatype (not initialized, empty, null)
bool8	bool (8 bit)
int8	signed int (8 bit)
uint8	unsigned int (8 bit)
int16	signed int (16 bit)
uint16	unsigned int (16 bit)
int32	signed int (32 bit)
uint32	unsigned int (32 bit)
int64	signed int (64 bit)
uint64	unsigned int (64 bit)
float, float32	float (32 bit)
double, float64	double (64 bit)
string	string (UTF-8)
timestamp	timestamp (FILETIME) 64 bit 100ns since 1.1.1601 (UTC)
flatbuffers	bytes as a complex data type encoded as a flatbuffer
raw	raw bytes

Arrays

type	description
arbool8	array of bool (8 bit)
arint8	array of signed int (8 bit)
aruint8	array of unsigned int (8 bit)
arint16	array of signed int (16 bit)
aruint16	array of unsigned int (16 bit)
arint32	array of signed int (32 bit)
aruint32	array of unsigned int (32 bit)
arint64	array of signed int (64 bit)
aruint64	array of unsigned int (64 bit)
arfloat, arfloat32	array of float (32 bit)
ardouble, arfloat64	array of float (64 bit)
arstring	array of string (UTF-8)
artimestamp	array of timestamps (FILETIME) 64 bit 100ns since 1.1.1601 (UTC)

Once the **Variable** node is initially created with it's configured *value type*, the written *value type* has to match the configured one exactly, otherwise an error code is returned to the client performing the write operation.

Built-in Flatbuffers types

To configure a node of a **built-in** flatbuffers type, set the *type* of the node to the address of the referred target type:

```
types/...
```

Example: Built-in flatbuffers

Flatbuffers **Variable** node of a **built-in** flatbuffers type:

```
nodes: [
    {
        ...
        "type": "types/datalayer/server-settings",
        "metadata": {
            "nodeClass": "Variable"
        }
    },
    ...
]
```

Custom Flatbuffers types

To register a **custom** flatbuffers, configure a node of *nodeClass* **Type** and add it to the *types* section.

- Set the *address* of the type to the address on which instance nodes should refer to it:

  types/...

Example:

```
types/inertial-value
```

- Set the *type* to the name of the corresponding compiled flatbuffers file (*.bfbs).
- Upload the *.bfbs file to appdata location:

  ```
  kvd/bfbs
  ```

Example: Custom flatbuffers

Type node of a **custom** flatbuffers type:

```
types: [
    {
        "address": "types/inertial-value",
        "type": "inertial-value.bfbs",
        "metadata": {
            "nodeClass": "Type"
        }
    },
    ...
]
```

A *Variable* node this type can be specified (value optional):

```
nodes: [
    {
        "address": "kvd/samples/variables/flatbuffers/inertial-value",
        "type": "types/inertial-value",
        "value": {
            "x": 1,
            "y": 2,
            "z": 3
        },
```

```
        "metadata": {
            "nodeClass": "Variable",
            "description": "This node is of the custom flatbuffers type
'types/inertial-value."
        }
    },
    ...
]
```

Configure Data Layer model nodes

Please see README-MODELS.md

Maintainers

See MAINTAINERS