DIGITAL TRANSFORMS PHYSICAL

UDD DRIVER DEVELOPMENT GUIDE AND DEEP DIVE

Solutions Consulting Team
Kepware Products

2023-06-16

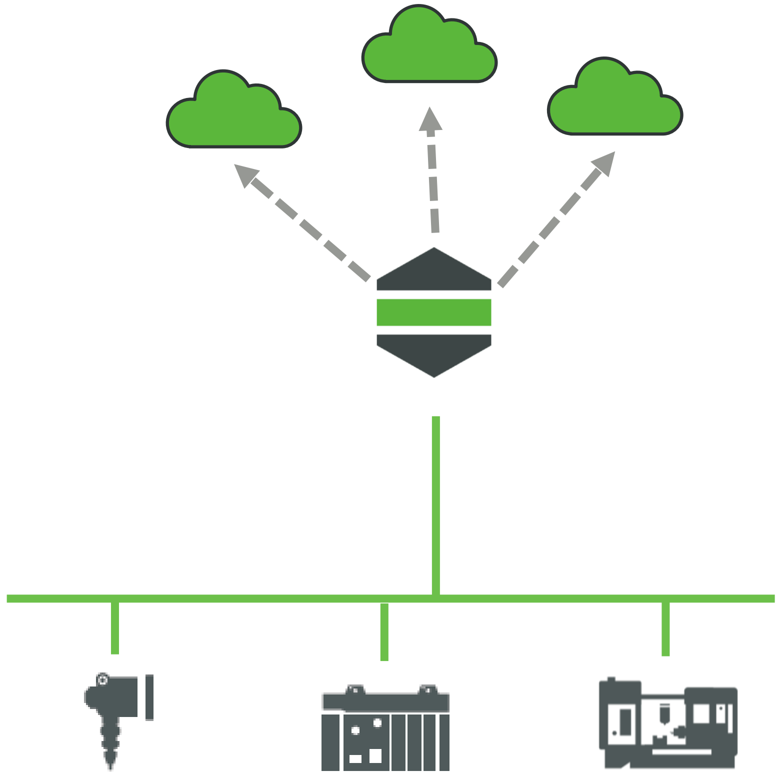
UNIVERSAL DEVICE DRIVER (UDD)

UNIVERSAL DEVICE DRIVER (SCRIPTABLE DRIVER)

Provide a flexible and dynamic solution to create custom driver profiles.
Establish greater connectivity to Tier 2 devices without native drivers.

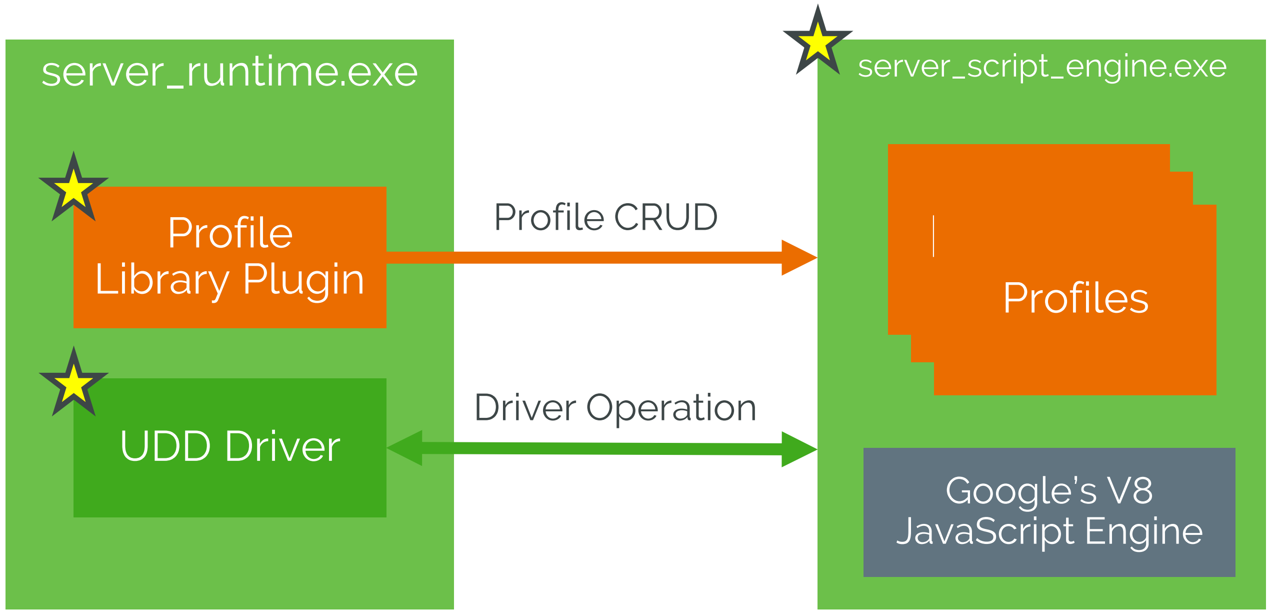
Accelerate Custom Driver Development:

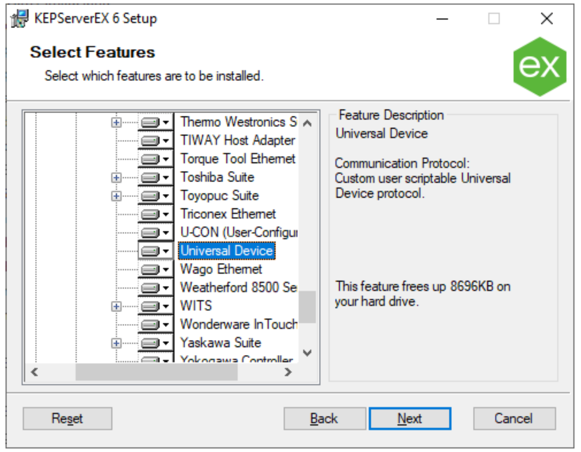
- Self-describing & generic
- Flexible & full control for the writer
- Solicited, Unsolicited & Mixed Ethernet profiles
  - Basic Driver Info (Comm Type & Custom properties)
  - Build & Parse Messages
- Targeted for technical/developer audience



OVERVIEW

- Universal Device Driver (UDD) allows users to write their own drivers using JavaScript
- Installing UDD adds three new components to Kepserver EX
  - Profile Library Plugin
  - UDD Driver
  - Script Engine Service
- UDD appears as a single item in the Installer under Drivers
- Installing UDD will also install the Plugin and Service




KEPServerEX 6 Setup

Select Features

Select which features are to be installed.

- Thermo Westronics S
- TIWAY Host Adapter
- Torque Tool Ethernet
- Toshiba Suite
- Toyopuc Suite
- Triconex Ethernet
- U-CON (User-Configu
- Universal Device
- Wago Ethernet
- Weatherford 8500 Se
- WITS
- Wonderware InTouch
- Yaskawa Suite
- Yokogawa Controller

Feature Description

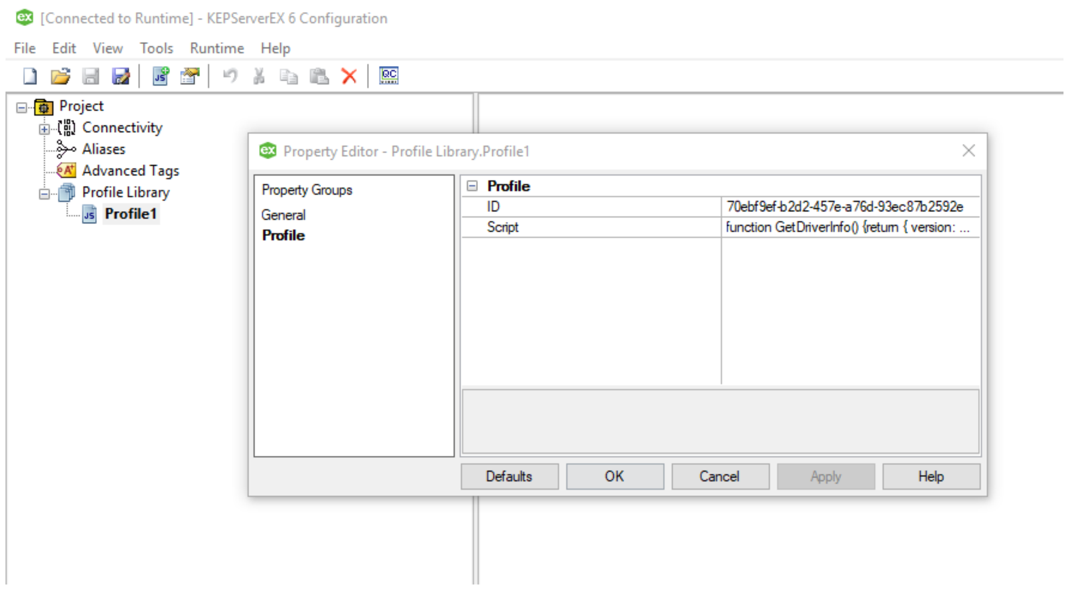
Universal Device

Communication Protocol:
Custom user scriptable Universal Device protocol.

This feature frees up 8696KB on your hard drive.

WHAT IS A PROFILE?

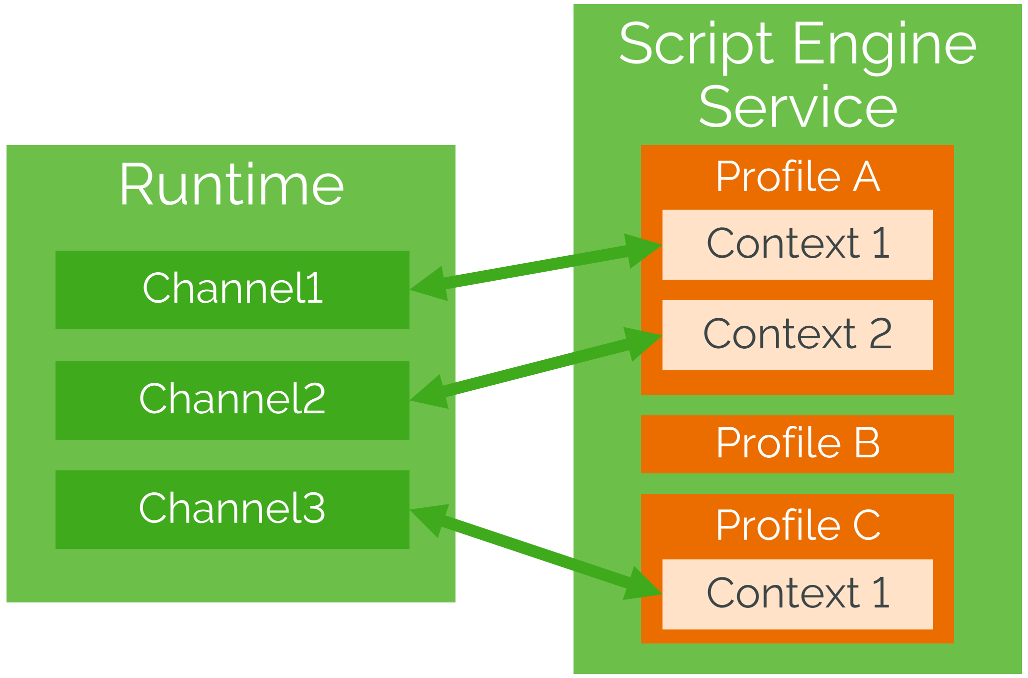
- A profile has two parts:
  - An ID (GUID)
  - A script (JavaScript)
- Each profile defines a device's behavior, just like a native Kepware driver
- The profile validates tags, builds payloads to send to the device and handles received payloads to complete tag transactions
- Think of each profile as a new type of driver



Profile	
ID	70ebf9ef-b2d2-457e-a76d-93ec87b2592e
Script	function GetDriverInfo() {return { version: ...

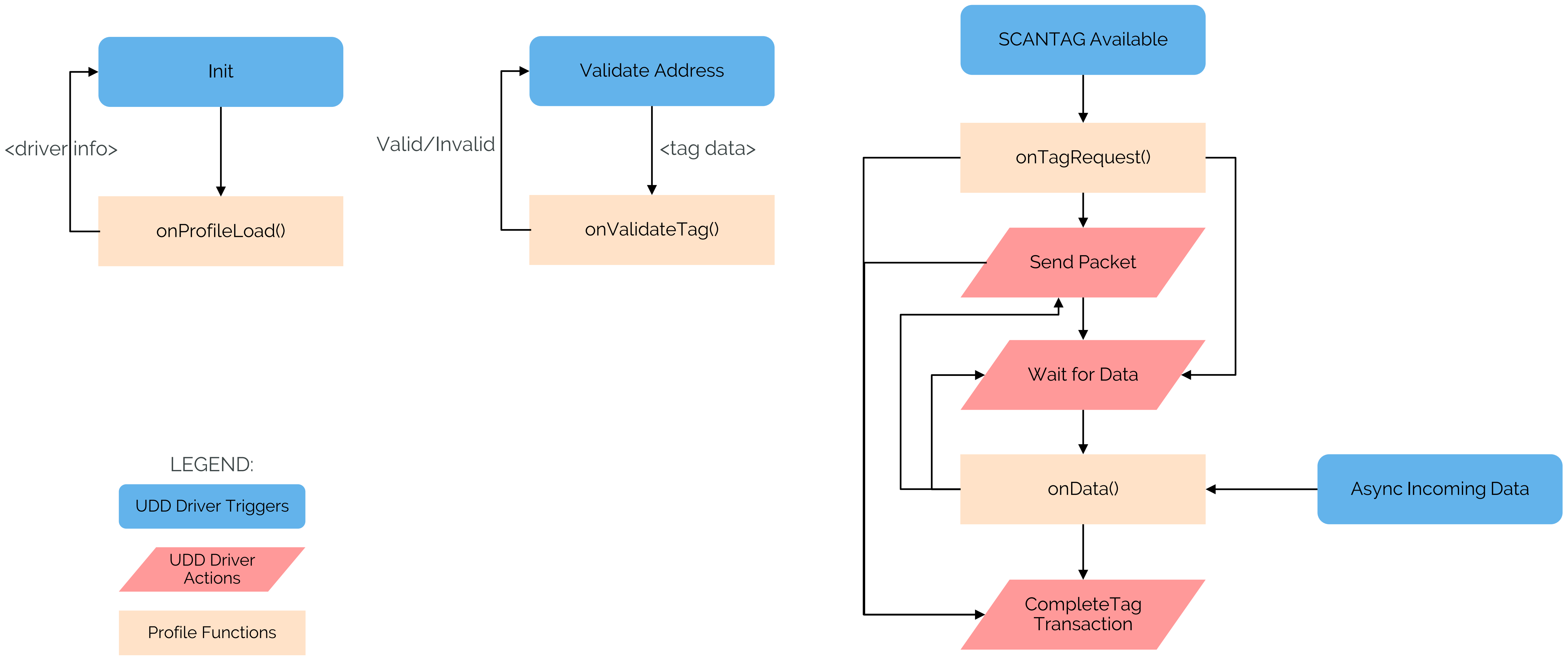
HOW IS A PROFILE USED?

- Each profile can be used by multiple channels (1:many)
- Each channel can only be linked to one profile (1:1)
- Each channel gets its own JavaScript context
  - Variables, caches and state created by one channel do not affect another channel, even if they use the same profile
- A profile is linked to a channel using the profile ID



PROFILE VERSION 2.0

UDD 2.0 DRIVER INTERNAL STATE MACHINE

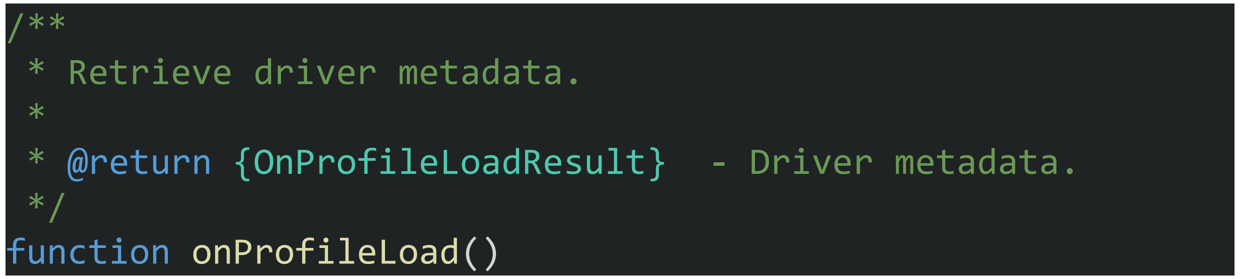


EVENT HANDLERS (FUNCTIONS) TO DEFINE

- onProfileLoad()
- onValidateTag()
- onTagsRequest()
- onData()

FUNCTION: ONPROFILELOAD()

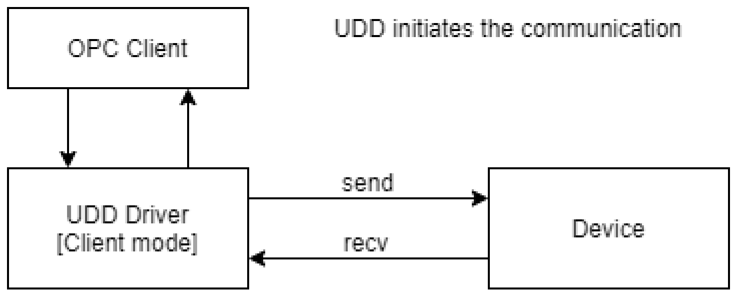
- Loaded by the driver instance initially to confirm UDD state machine version and socket control mode to use.
    - Used to initialize any supporting variables, cache or other elements of the profile
- Return profile configuration results (**OnProfileLoadResult**) and any properties to configure the driver.
    - Property **version** identifies the profile version to use. Currently “2.0”
    - Property **mode** defines which mode the driver instance should operate in
        - Essentially socket control between driver and the device
- Type Definition:
    - OnProfileLoadResult



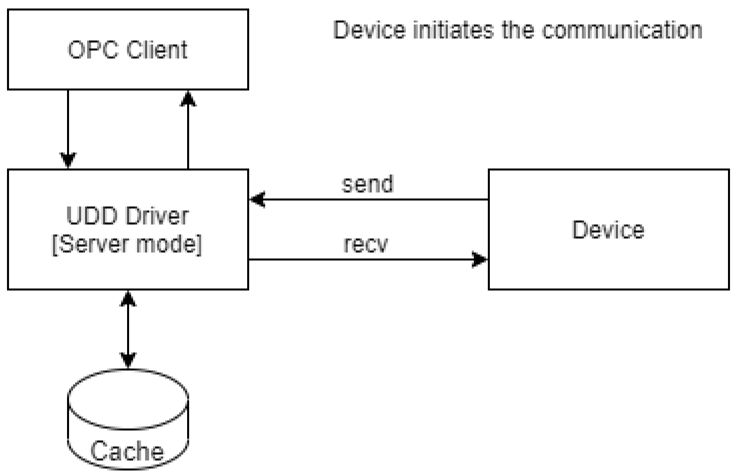
```
/**
 * Retrieve driver metadata.
 *
 * @return {OnProfileLoadResult}  - Driver metadata.
 */
function onProfileLoad()
```

MODE PARAMETER SOCKET CONTROL

Client mode:



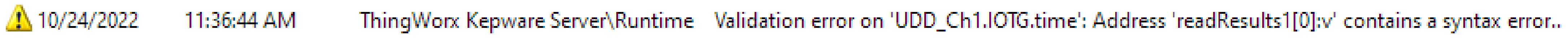
Server mode:



- For TCP transport, **mode** will determine whether the driver instance will send the connection request (client) or listen the connection request (server)
  - **NOTE:** Driver does not execute Tag transaction event handlers until a TCP socket has been established
- For UDP transport, **mode** doesn't affect behavior since it is "connectionless".
  - The listening port will be established based on the device parameters in the driver configuration

FUNCTION: ONVALIDATETAG()

- Driver calls function for each tag referenced
  - static or dynamic tag references from clients or plug-ins of Kepware
- Receives input argument object (**info**) with tag information to validate
- Apply logic or REGEX to check if Tag created is valid for your script
  - Correct tag properties like data type, or read/write access
- Return validation results (**OnValidateTagResult**) and any properties to modify if needed.
  - If invalid, server displays “tag address invalid” error just like other commercially-created drivers



```
10/24/2022    11:36:44 AM    ThingWorx Kepware Server\Runtime    Validation error on 'UDD_Ch1.IOTG.time': Address 'readResults1[0]:v' contains a syntax error..
```

FUNCTION: ONVALIDATETAG() – INPUT/OUPUT

- Input argument:
  - **info** is an object with a single member – **tag**
  - **Tag** is an object with the following properties:
    - **address, dataType, readOnly, bulkId**
- Return object:
  - **OnValidateTagResult** is returned with various parameters to modify the tag properties
  - A Boolean value for **valid** will indicate whether the tag is valid (true) or not (false)
  - A number value for **bulkId** will identify the bulkId group for the tag that is being validated.

```
/**
 * Validate an address.
 *
 * @param {object}  info          - Object containing the function arguments.
 * @param {Tag}     info.tag      - Single tag.
 *
 * @return {OnValidateTagResult}  - Single tag with a populated '.valid' field set.
 *
 * */
```

- Type definitions:
  - Tag
  - OnValidateTagResult

FUNCTION: ONVALIDATETAG() – STRATEGIES

- Understand the tag addressing for the device/protocol and plan accordingly
  - Scope of data/addresses in the application protocol will help inform the best approach to validate tags
  - Protocols with numeric addressing (MODBUS for example) may only need simple comparison logic
    - Example: If Modbus tag address is between 400001 and 465535 – Holding Register
  - Protocols that may leverage string/alphanumeric tag addresses are good scenarios to leverage REGEX comparisons to validate addresses.
  - Protocols that have a fixed list of addresses or commands could be validated by using a fixed list of addresses in the profile
- Remember that **tag.address** property is always a string! Make sure to handle numeric only addresses appropriately (i.e. convert to number as need during checks)
- Plan to handle "default" data types in case a tag is created with "default" data type

FUNCTION: ONVALIDATETAG() – USING BULK TAG GROUPS

- Bulk Tag groups allow for transactions to update multiple tags from a single request of the device
  - Common use cases include:
    - requests that return JSON objects with multiple values
    - protocols that can block requests for multiple registers like Modbus
- If a profile will use a bulk tag group, it will need to assign a **bulkId** for every tag used. This means even a single tag with a unique request that provides only its value must be assigned its own **bulkId** during tag validation.

REGEX EXAMPLE USED IN HTTP CLIENT SAMPLE CODE

Permitted Example Address: myObject[3]:memberKey

/.. / -- open & close; indicates the start & end of regular expression

^ -- matches beginning of the string

[..] -- character set match; match any char in set

a-zA-Z -- set; matches a char in range a to z, case sensitive

+ -- quantifier; match 1 or more of the preceding

(..)-- capturing group; groups multiple tokens together and creates a capture group for extracting a substring

\[-- escaped character; matches a "[" character

[..] – char set match

0-9 -- set; matches a char 0 to 9

+ -- quantifier; match 1 or more of the preceding

\] – escaped char; matches a "]" character

| -- alternate; acts like a Boolean OR. Matches expression before or after the "|"

: -- character; matches a ":" character

[..] -- character set match; match any char in set

a-zA-Z -- set; matches a char in range a to z, case sensitive

+ -- quantifier; match 1 or more of the preceding

* -- quantifier; match 0 or more of the preceding

$ -- end of string

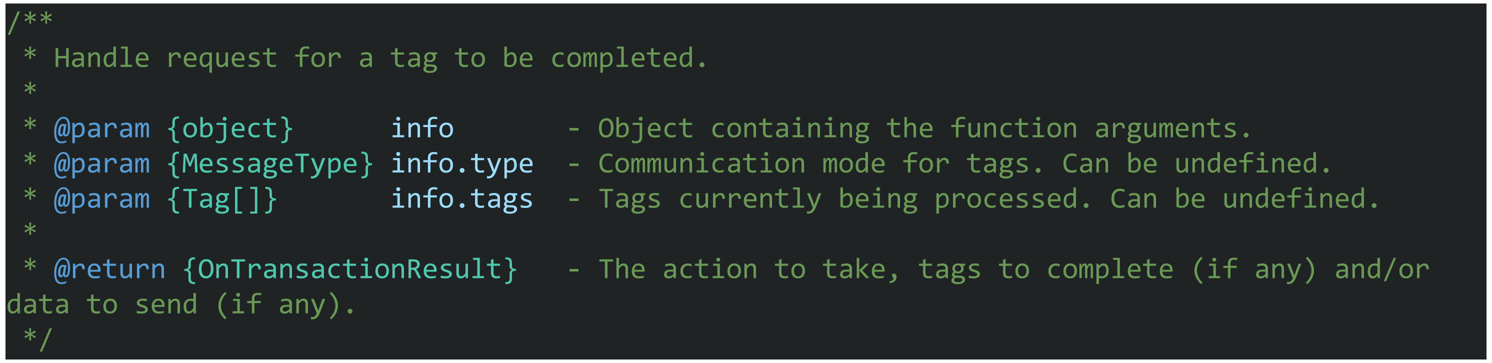
```
/*
     * The regular expression to compare address to.
     * ^, & Starting and ending anchors respectively. The match must occur between the
     * two anchors
     * [a-zA-Z]+ At least 1 or more characters between 'a' and 'z' or 'A' and 'Z'
     * [0-9]+ At least 1 or more digits between 0 and 9
     * | is an or statement between the expressions in the group
     * ()* Whatever is in the parentheses can appear 0 or unlimited times
     */
    let regex = /^[a-zA-Z]+(\[[0-9]+\]|:[a-zA-Z]+)*$/;

    try {
        // Validate the address against the regular expression
        if (regex.test(info.tag.address)) {
            info.tag.valid = true;
            // Fix the data type to the correct one
            if (info.tag.dataType === data_types.DEFAULT){
                info.tag.dataType = data_types.STRING
            }
            log('onValidateTag - address "' + info.tag.address + '" is valid.’,
               VERBOSE_LOGGING)
        } else {
            info.tag.valid = false;
            log("ERROR: Tag address '" + info.tag.address + "' is not valid");
        }
        return info.tag
    }
    catch (e) {
        // Use log to provide helpful information that can assist with error resolution
        log("ERROR: onValidateTag - Unexpected error: " + e.message);
        info.tag.valid = false;
        return info.tag;
    }
```

FUNCTION: ONTAGSREQUEST()

- Driver calls function for actions requested from server
  - Start of a Tag transaction
  - Possible requests: Read or Write requests
- Receives input argument object (**info**) with various information about the request
  - Property **type** provides information about the request type from the server (read or write)
    - **NOTE: Write requests types currently only support one tag write per transaction.**
  - Property **tags** provides an array of **tag** objects associated with the request
    - **NOTE: Will include all tags that are associated with the bulk group for the current transaction.**
- Return information (**OnTransactionResult**) to determine next state machine action for the driver to take
  - Should the driver send a message to the device?
  - Is a cached value returned for the tag and complete the transaction?
  - Action will be determined based on behaviour definitions of protocol

FUNCTION: ONTAGSREQUEST() – INPUT/OUTPUT

- Input argument:
  - **info** is an object with a two members – **type** and **tags**
  - **Tag** object will contain a **value** property for a "write" transaction type
    - This would be the "write" value received from a client application (ex: OPC client) by the server to write to the device
- Return object:
  - **OnTransactionResult** is returned with various parameters determine next step to process tag transaction.
- Type definitions:
  - MessageType
  - Tag
  - OnTransactionResult



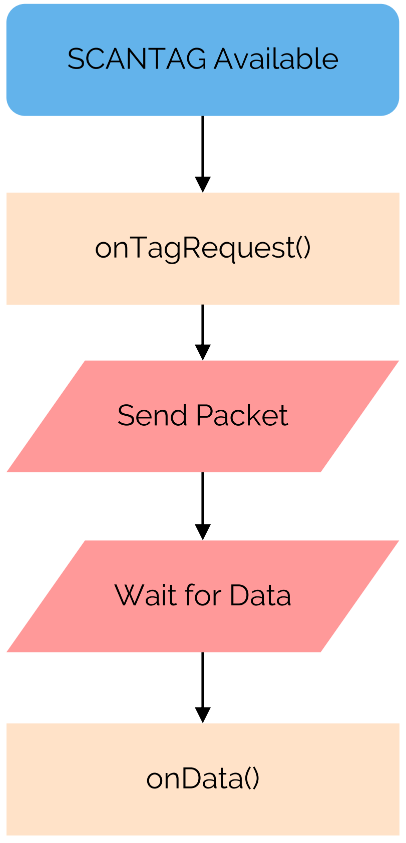
```
/**
 * Handle request for a tag to be completed.
 *
 * @param {object}      info       - Object containing the function arguments.
 * @param {MessageType} info.type  - Communication mode for tags. Can be undefined.
 * @param {Tag[]}       info.tags  - Tags currently being processed. Can be undefined.
 *
 * @return {OnTransactionResult}   - The action to take, tags to complete (if any) and/or
data to send (if any).
 */
```

FUNCTION: ONTAGSREQUEST() – TRANSACTION RESULTS

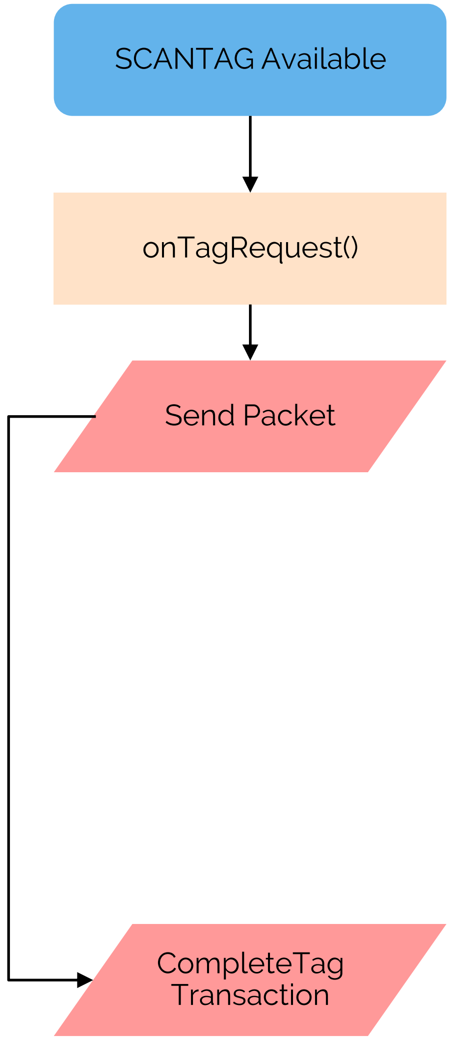
- **OnTransactionResult** has three properties to return that are used to command the driver on next steps
  - **action**: provide direction on next steps to take
    - receive – transaction needs to receive data from the device
    - complete – transaction is complete
    - fail – transaction has serious failure and provides a “Device Not Responding” type result to server
  - **tags**: (optional) provide array of tag information when transaction has been completed
    - Used to provide a value that is returned to the server as a response to the transaction
  - **data**: (optional) message of the raw data to be sent to a device
    - Byte array format
    - If populated, will always send a message in a receive or complete action type.

FUNCTION: ONTAGSREQUEST() – TRANSACTION RESULTS

- Use Case Examples:
  1. Send read/write request message with expect response from device
     - Return object – {action: "Receive", data: [binary array]}
     - Driver will send message
     - Transition to a "wait for data" state
  2. Send write request message without response from device
     - Return object – {action: "Complete", data: [binary array]}
     - Driver will send message
     - Transition to "Transaction Complete" state



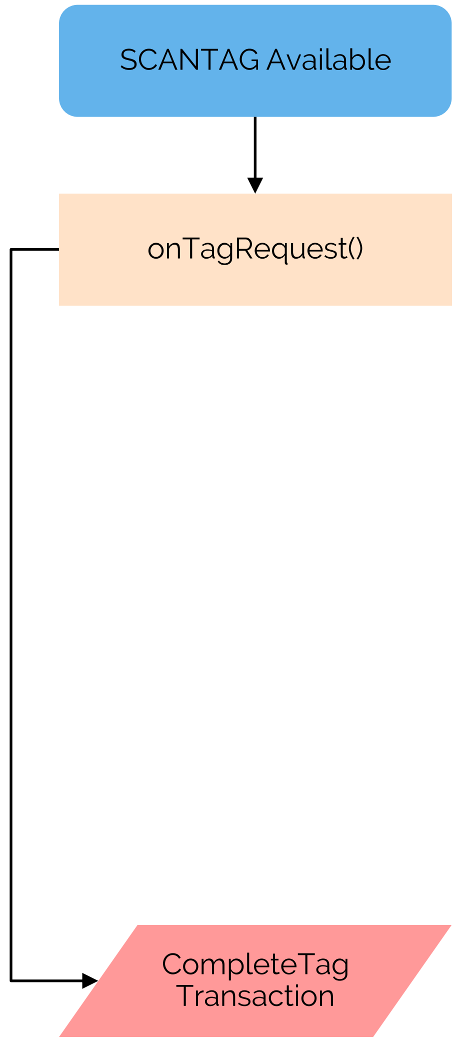
Use Case 1



Use Case 2

FUNCTION: ONTAGSREQUEST() – TRANSACTION RESULTS

- Use Case Examples:
  3. Read request with value for tag stored in cache
     - Typically used with unsolicited data
     - Return object – {action: "Complete", tags: [tagObjectwithValue]}
     - Provide tag object with an updated value
     - Transition to "Transaction Complete" state



Use Case 3

FUNCTION: ONTAGSREQUEST() – STRATEGIES

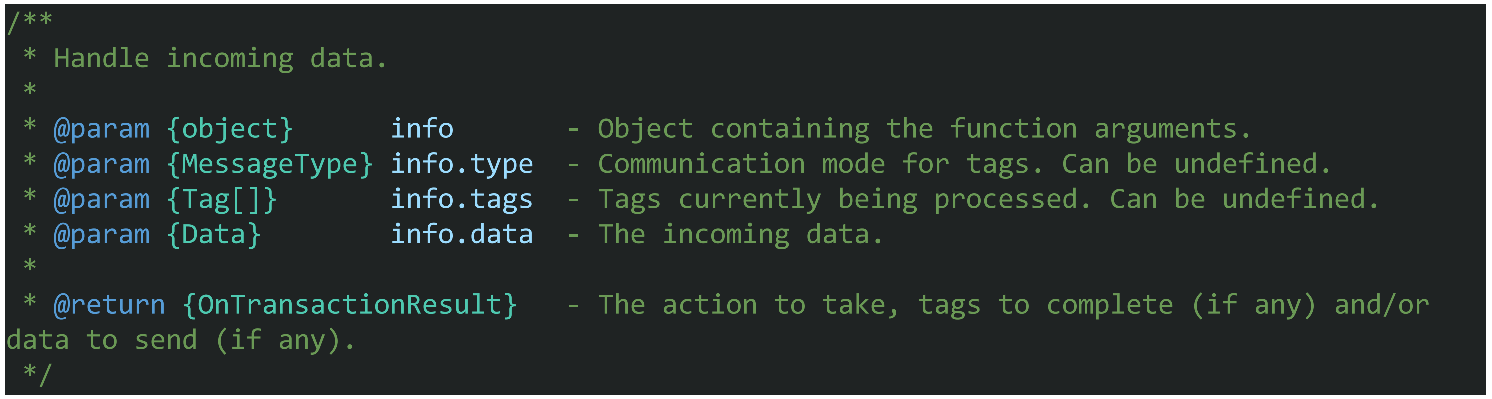
- If the Tag transaction is not completed in this request (ex: cases 1 and 2) the transaction is considered a pending action.
  - Future messages received by the driver from the device will have this Tag transaction associated with the response.

FUNCTION: ONDATA()

- Driver calls function when data is received; Processes received data
  - Either a response to a request or unsolicited message received from device
- Receives input argument object (**info**) with various information about the received message
  - Property **type** provides information about the request type from the server (read or write)
  - Property **tags** provides an array of **tag** objects associated with the request
    - **NOTE: Will include all tags that are associated with the bulk group for the current transaction.**
  - Property **data** provides an byte array of the raw data received from the device
- Return information (**OnTransactionResult**) to determine next state machine action for the driver to take
  - Has the driver received a complete message from the device?
  - Should the driver send an acknowledge message to the device?
  - Is a value returned for the tag and complete the transaction?

FUNCTION: ONDATA() – INPUT/OUTPUT

- Input argument:
  - **info** is an object with a three members – **type, tags** and **data**
  - **NOTE: type** and **tags** are only provided if an outstanding Tag transaction is pending completion
- Return object:
  - **OnTransactionResult** is returned with various parameters determine next step to process tag transaction.
- Type definitions:
  - MessageType
  - Tag
  - OnTransactionResult
  - Data



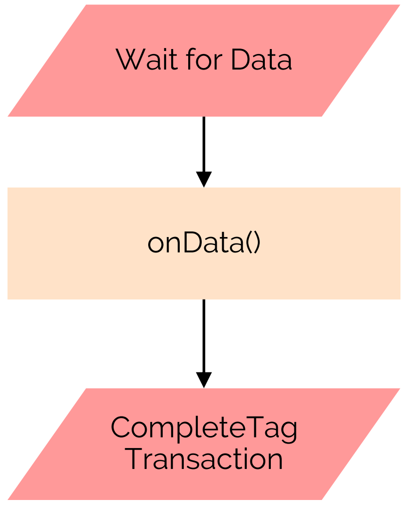
```
/**
 * Handle incoming data.
 *
 * @param {object}      info       - Object containing the function arguments.
 * @param {MessageType} info.type  - Communication mode for tags. Can be undefined.
 * @param {Tag[]}       info.tags  - Tags currently being processed. Can be undefined.
 * @param {Data}        info.data  - The incoming data.
 *
 * @return {OnTransactionResult}   - The action to take, tags to complete (if any) and/or
data to send (if any).
 */
```

FUNCTION: ONDATA() – TRANSACTION RESULTS

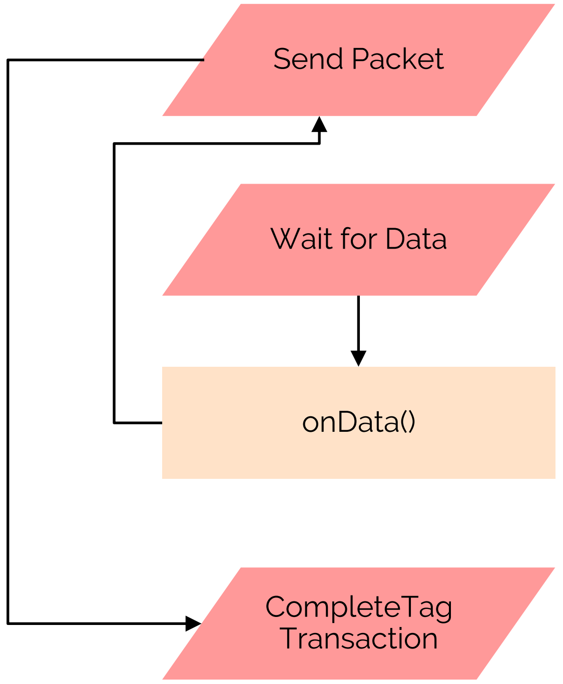
- **OnTransactionResult** has three properties to return that are used to command the driver on next steps
  - **action**: provide direction on next steps to take
    - receive – transaction needs to receive data from the device
    - complete – transaction is complete
    - fail – transaction has serious failure and provides a "Device Not Responding" type result to server
  - **tags**: (optional) provide array of tag information when transaction has been completed
    - Used to provide a value or quality that is returned to the server as a response to the transaction
  - **data**: (optional) message of the raw data to be sent to a device
    - Byte array format
    - If populated, will always send a message in a receive or complete action type.

FUNCTION: ONDATA() – TRANSACTION RESULTS

- Use Case Examples:
  1. Receive response from device as a result of a read/write request
     - Return object – {action: "Complete", tags: [updated tag with value (if read)]}
     - Transition to "Transaction Complete" state
  2. Receive response from device as a result of a read/write request and send acknowledgement message back
     - Return object – {action: "Complete", tags: [updated tag with value (if read)], data: [binary array]}
     - Transition to "Send Packet" then "Transaction Complete" state



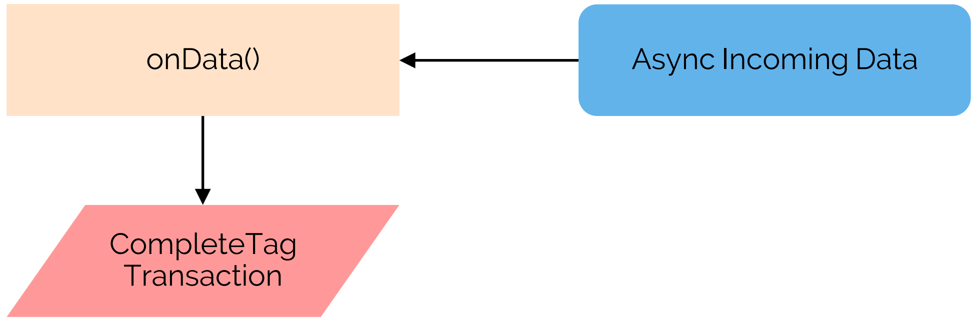
Use Case 1



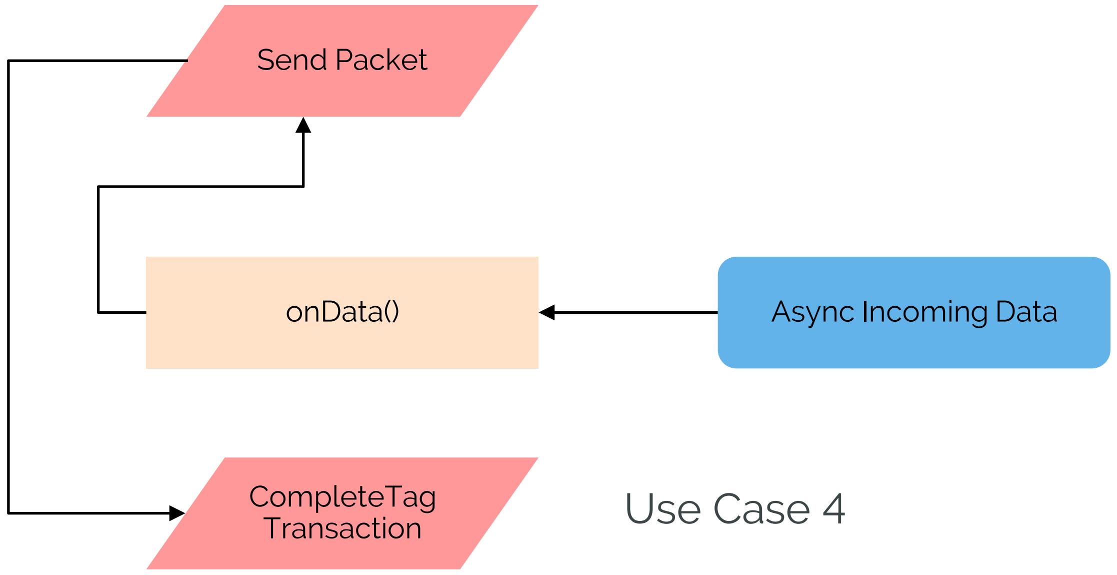
Use Case 2

FUNCTION: ONDATA() – TRANSACTION RESULTS

- Use Case Examples:
  3. Receive unsolicited message with values for tags to store in cache
     - Return object – {action: "Complete}
     - Update any cached values necessary. Tag gets updated in different Tag transaction (see case 3)
     - Transition to "Transaction Complete" state
  4. Receive unsolicited message with values for tags to store in cache and send acknowledgement message back
     - Return object – {action: "Complete, data: [binary array]}
     - Update any cached values necessary. Tag gets updated in different Tag transaction (see case 3)
     - Transition to "Send Packet" then "Transaction Complete" state



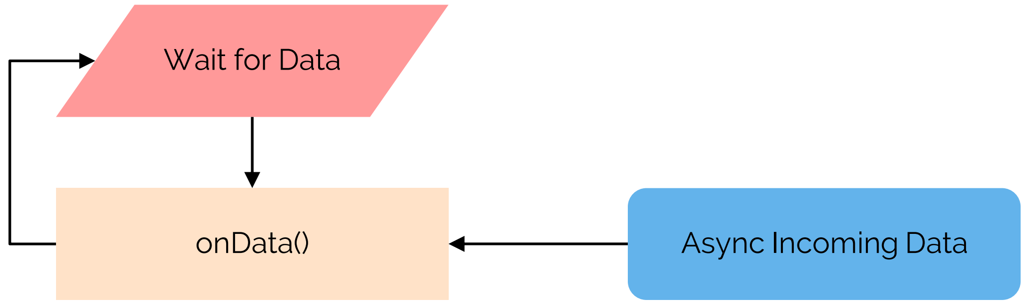
Use Case 3



Use Case 4

FUNCTION: ONDATA() – TRANSACTION RESULTS

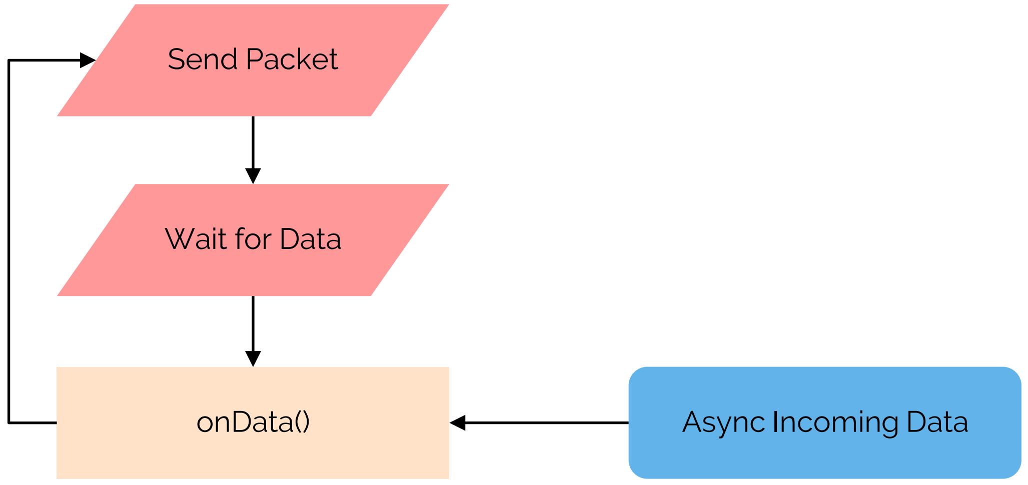
- Use Case Examples:
  5. Receive response (unsolicited or request response) but response is incomplete
     - Return object – {action: “Receive”}
     - Transition to “Wait for Data” state
     - Complete as necessary once response is complete from device



Use Case 5

FUNCTION: ONDATA() – TRANSACTION RESULTS

- Use Case Examples:
  6. Multiple message transactions to/from device
     - Started either from a request from driver or from unsolicited message from device
     - Return object – {action: “Receive”, data: [binary array]}
     - Transition to “Send Packet” as many times as needed
     - Complete as necessary once message transactions are complete to/from device



Use Case 6

FUNCTION: ONDATA() – STRATEGIES

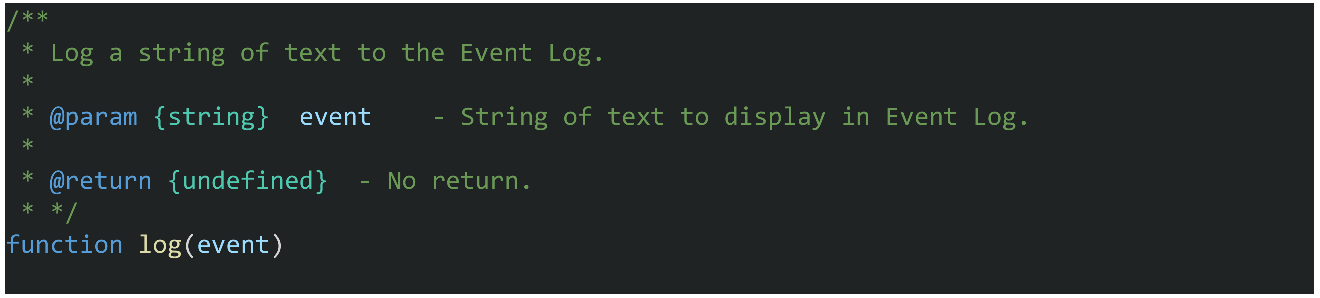
- If a Tag transaction is pending completion, it will always be referenced in the **onData** input object (**info**).
  - **Mixed Mode protocol handling**: Need to validate if received message is response to tag transaction (request message) or an unsolicited message
  - If an unsolicited message is received from a device, prior to the response of a solicited message **info** will include an unsolicited data response and have a tag transaction present.
    - Process unsolicited message as needed, then transition state machine back to "Waiting for Data" state (ex: use case 5)
- Ensure to develop a strategy to properly check if the received data has one or more application protocol messages
  - **Mixed Mode protocol handling**: The core UDD driver may receive multiple packets over the wire quick enough that the **onData** input could include multiple application protocol messages to process in one **onData** call.

COMPLETETAG: QUALITY PROPERTY (V6.14 AND NEWER)

- Allows the profile to assign tag qualities of **“Good”**, **“Bad”**, or **“Uncertain”**.
- Necessary for bulk tag groups that may have mixed responses from the device.
  - Example: Response that has good values for all but one tag, the single tag can be flagged with “Bad” quality while the others are updated with “Good” quality.
- Guidance when assigning the quality property:
  - If a tag value **is** provided, quality will be **assumed “Good”** unless the quality property is assigned a different value.
  - If a tag value **is not** provided, quality will default to **“Bad”** regardless if a quality property is assigned.
  - A tag value **is required** if quality is assigned **“Uncertain”**. If no value is provided, the **tag transaction will fail for all tags associated with the transaction**.

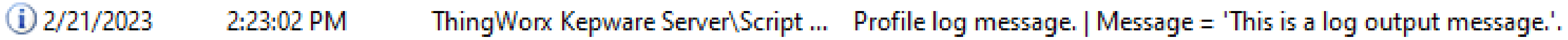
SUPPORTING FUNCTIONS

- Use **log()** will log a string of text into the Kepware Event Log
  - Use this to provide helpful errors/information about the communications
  - Mechanism to debug during profile development



```
/**
 * Log a string of text to the Event Log.
 *
 * @param {string}  event    - String of text to display in Event Log.
 *
 * @return {undefined}  - No return.
 * */
function log(event)
```

- Example Output:



```
2/21/2023    2:23:02 PM    ThingWorx Kepware Server\Script ...   Profile log message. | Message = 'This is a log output message.'.
```

SUPPORTING FUNCTIONS

- A built-in **cache** is available for each driver instance to store data between transactions
  - Particularly useful for unsolicited transactions
  - **initializeCache()** to initialize the storage. Typically done during **onProfileLoad**.
  - **writeToCache()** to add or update stored data in an instance specific cache
  - **readFromCache()** to read data from an instance specific cache.
    - Type Definition: [CacheReturn]()

```
/**
 * Initialize a cache to store data between transactions.
 *
 * @param {number}  maxSize    - (optional) Size of cache to set. (10k maximum)
 *
 * @return {undefined}  - No return.
 * */
function initializeCache(maxSize)

/**
 * Write or update a value to instance specific cache.
 *
 * @param {string}  key    -  Address of the cache item to reference. Often the tag
 *                              address
 * @param {*}  value    -  Value of the tag. Max length of 4096 characters
 *
 * @return {string}  - Return “success” or “error” based on success
 * */
function writeToCache(key, value)

/**
 * Read a value from the instance specific cache.
 *
 * @param {string}  key    -  Address of the cache item to reference. Often the tag
 *                              address
 *
 * @return {CacheReturn}  - Object containing the stored value in the instance specific
 *                              cache.
 * */
function readFromCache(key)
```

UDD TECH SPECS

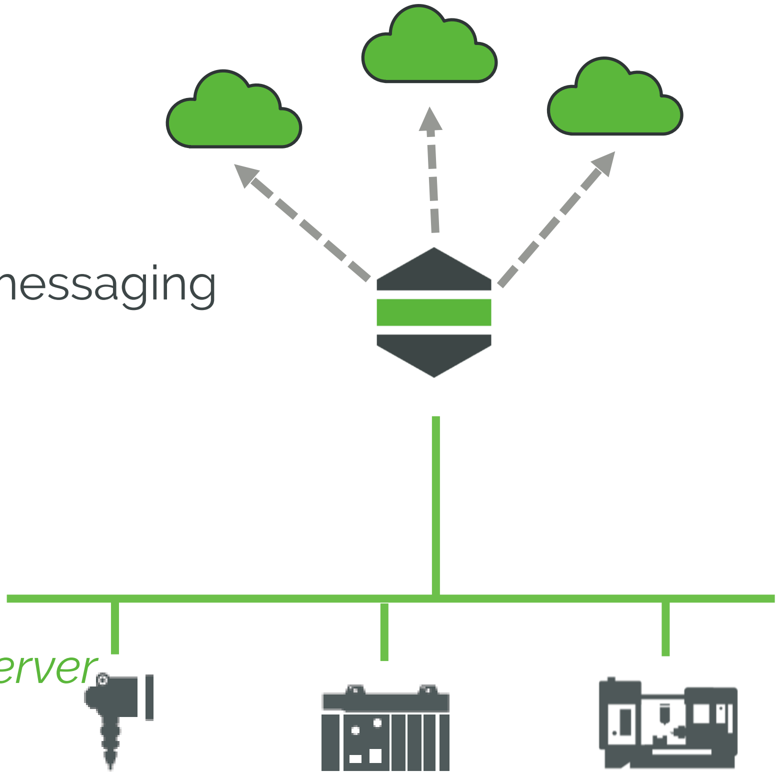
UNIVERSAL DEVICE DRIVER VERSION 2.0

Mixed Mode, Publish / Subscribe and Unsolicited Profile Support

Numerous devices in the field require flexible interactions to transmit data. The latest enhancements to UDD now allow the script writer to leverage Kepware as a listener when connecting to downstream devices.

Key Updates:

- Profile messaging architecture adjusted for event-based commands
- **Mixed-mode** can be designed to where the solicited and unsolicited messaging can be supported within the same profile
- Additional configurable timing properties & performance updates



**Available in select packages with KEPServerEX and ThingWorx Kepware Server*

DRIVER SUMMARY – UDD VERSION 2.0

- Solicited/Unsolicited/Mixed/Pub-Sub Ethernet protocols using TCP or UDP
- One device per channel
- Kepware v6.14 and newer adds bulk tag processing (i.e. multiple tags per Tag transaction to process)
- Quality properties can be explicitly controlled (v6.14 and newer)
- No Date or Array (native) datatypes
  - Use strings to hold these types as needed
- 1024 Channels

RELATED RESOURCES

- Universal Device Driver (UDD) Product Page
- Configuring Universal Device Driver and Profile Library Guide
- PTC's Github UDD Example Repository: PTCInc/Universal-Device-Driver-Examples (github.com)
- Sample codes installed under the Kepware installation folder:
  - C:\Program Files (x86)\PTC\Thingworx Kepware Server 6\Examples\Universal Device Sample Profiles
  - C:\Program Files (x86)\Kepware\KEPServerEX 6\Examples\Universal Device Sample Profiles

TYPEDEF REFERENCES

TYPEDEF: MESSAGETYPE, DATATYPE, DATA

```
@typedef {string} MessageType - Type of communication "Read",
"Write".

@typedef {string} DataType - Kepware datatype "Default",
"String", "Boolean", "Char", "Byte", "Short", "Word", "Long",
"DWord", "Float", "Double", "BCD", "LBCD", "Date", "LLong",
"QWord".

@typedef {number[]} Data - Array of data bytes. Uint8 byte
array.
```

TYPEDEF: ONPROFILELOADRESULT

```
@typedef {object}  OnProfileLoadResult
@property {string} version - Version of the driver.
@property {string} mode    - Operation mode of the driver/socket
"Client“ or "Server".
```

TYPEDEF: TAG

```
@typedef {object}   Tag
@property {string}   Tag.address  - Tag address.
@property {DataType} Tag.dataType - Kepware data type.
@property {boolean}  Tag.readOnly - Indicates permitted
communication mode.
@property {number}  Tag.bulkId   - (optional) Integer that
identifies the group into which to bulk the tag with other tags.
@property {*}        Tag.value   - (optional) Desired value of
the tag. This field is only populated when a MessageType is
Write in a transaction.
```

TYPEDEF: COMPLETETAG

```
@typedef {object}  CompleteTag
@property {string} Tag.address   - Tag address.
@property {*}      Tag.value     - (optional) Tag value.
@property {string}  Tag.quality   - (optional) Quality of Tag:
"Good", "Bad", "Uncertain"
```

TYPEDEF: ONPROFILELOADRESULT

```
@typedef {object}  OnProfileLoadResult
@property {string} version     - Version of the driver.
@property {string} mode        - Operation mode of the driver
"Client", "Server".
```

TYPEDEF: ONVALIDATETAGRESULT

```
@typedef {object}   OnValidateTagResult
@property {string}  address      - (optional) Fixed up tag
address.
@property {DataType} dataType     - (optional) Fixed up Kepware
data type. Required if input dataType is "Default".
@property {boolean} readOnly     - (optional) Fixed up permitted
communication mode.
@property {number}  bulkId       - (optional) Integer that
identifies the group into which to bulk the tag with other tags.
Universal Device Driver assigns the next available bulkId, if
undefined. If defined for one tag, must define for all tags.
@property {boolean} valid        - Indicates address validity.
```

TYPEDEF: ONTRANSACTIONRESULT

```
@typedef {object}  OnTransactionResult
@property {string} action - Action of the operation:
"Complete", "Receive", "Fail".
@property {CompleteTag[]} tags   - Array of tags (if any active)
to complete. Undefined indicates tag is not complete.
@property {Data}   data   - The resulting data (if any) to
send. Undefined indicates no data to send.
```

TYPEDEF: CACHERETURN

```
@typedef {object}   CacheReturn
@property {string}  key - Key associated with the value returned
@property {*}       value - Stored value in cache associated
with the key. Maximum 4096 characters.
```

ptc

DIGITAL TRANSFORMS PHYSICAL

THANK YOU

ptc.com