Profile Library Plug-In

Table of Contents

Profile Library Plug-In

Help version 1.034

CONTENTS

Overview

The Profile Library Plug-In allows a user to create script-based profiles to use in conjunction with the Universal Device Driver to communicate with a wide variety of Ethernet devices or to operate on files. Make use of the Profile Library Plug-In to implement custom profiles in cases when there is no native driver for a particular protocol or device or in situations requiring file access. The Profile Library Plug-In offers the following features:

- Ability to customize profiles to meet specific connectivity needs
- Script-based interface which gives users flexibility in what functionality to implement
- Ability to edit a profile once and push edits to all instances of that profile

Architecture

The Profile Library Plug-in is used to configure and maintain profiles that are consumed by the Universal Device Driver. These profiles contain a script that implements business logic for device communication or for operating on files. The Ethernet (Server or Client) Mode profile script defines the interface and implementation required for a specific protocol that the device uses to communicate. The File Mode profile script defines the interface and implementation required for operating on files.

Universal Device Driver channels require a linked profile that must be assigned during channel creation. This process is called linking the profile to the channel. Once a profile is linked and a device is created on the channel, the profile's script is registered with the server's script engine, which prepares the server and driver for communication using the profile's script. At this point all communication happens between the script engine, driver, and device / file.

If the profile is updated or modified, the active script is un-registered and the updated script from the modified profile is registered with the script engine. Once this process is complete, all driver communication uses the updated script.

Important Concepts

- **Client / Server**: This references the type of Ethernet connection established with a device. In client mode, a TCP/IP socket connection is initiated by the Universal Device Driver to the specified IP address using the defined port number. In server mode, the Universal Device Driver listens for an incoming connection on the specified IP address and port number.
- **Solicited**: This refers to a communication mode in which the Universal Device Driver requests data from a device. UDP is also available.
- **Unsolicited**: This refers to a communication mode in which data is sent to the Universal Device Driver from a device without any corresponding request.
- **Data Cache**: A built-in cache library is provided to enable asynchronous communications, allowing the script to internally store tag data until it is needed for a read or write operation.
- **Transaction**: A discrete set of I/O operations with a device to perform some action, such as reading or writing data, starting a subscription, or completing a handshake.
- **Completing a Tag**: Tags can be Read or Write and need to be "completed" by the driver. When tags are requested, it is the responsibility of the script to perform the necessary steps to complete that request. Only when active tags are completed (successfully or unsuccessfully) are the next tags requested.
- **Bulk Tags**: Tags that share the same bulkId are blocked together and provided to the onTagsRequest, onData, onFileTagsRequest, onFileOperations, and onFileChange functions, allowing the script to complete all the tags in the block. This feature is typically used to improve driver efficiency. ● *See Bulk Tags section for more information.*
- **Array Tags**: Tags that store an array of values with the same data type, in a single tag. Only single dimension arrays are supported. The number of elements in the array is determined by the arrayColumns tag attribute, set in onValidateTag. Array tags are not blocked with other tags like Bulk Tags above. Only one array tag is provided to onTagsRequest, onData, onFileTagsRequest, onFileOperations, and onFileChange functions. ● *See Array Tags section for more information.*
- **Profile Inputs**: Values that can be passed to the script via the onProfileInputsChange function. This function is called after any of the profile input device properties are changed by the user. *See Profile Inputs section for more information.*

🔹 *See Also*: *Profile Type*, *Bulk Tags*, *Array Tags*, *Profile Inputs*

Profile Types

The Universal Device Driver provides a flexible framework for implementing communication protocols or for operating on files.

Ethernet profiles implement communication protocols of which most types fall into one of the following four broad categories, independent of the underlying transport mechanism. Each category provides general design guidance, but choosing one does not restrict its functionality.

- **Simple Solicited**: In Simple Solicited, tags are fulfilled using synchronous data requests as needed. No messages are expected from the device when there is no outstanding request.
- **Simple Unsolicited**: In Simple Unsolicited, the script never requests data. The device determines what data to send and when to send it. Tags are fulfilled from a cache populated by the script as data becomes available.
- **Mixed Mode**: In Mixed Mode, some tags are fulfilled synchronously and others populated asynchronously. The script determines that based on user-defined criteria for how to process each tag request.
- **Pub / Sub**: In Pub / Sub, tags are fulfilled asynchronously only after a request is made to the device that enables data publication. The script must have a mechanism for tracking if a subscription already exists for tags as they are requested.

File profiles implement file operations such as:

- Open file
- Close file
- Read file
- Write file
- Read line
- Write line
- Create file
- Move File
- Delete file
- Asynchronous watch of a directory
- Stop Asynchronous watch of a directory

🌟 **Tip**: Examples and templates can be found in the install directory \<product>\Examples\Universal Device Sample Profiles and at https://github.com/PTCInc/Universal-Device-Driver-Examples.

Creating and Configuring Profiles

The Profile Library Plug-In provides the ability to create profiles via the server configuration interface and the Configuration API Service.

Note: For communication to occur, a channel must be linked to a valid profile.

What is a profile?

A Profile is a collection of properties that together provide all the information that the Universal Device Driver needs to communicate with a device or to operate on a file. Properties on the profile include the Name, Description, ID, and Script. Of these the script and ID are the most important.

The **Ethernet Mode** profile script defines all the instructions required by the driver to communicate over a specific protocol with a device.

The **File Mode** script defines all the instructions required by the driver to operate on a file or files.

The Ethernet and File Mode script interfaces are defined in more depth in the script section.

Tip: The profile ID property is a unique identifier that is used to link the profile to a Universal Device Driver channel and is in a GUID format.

Profile Properties — General

These profile properties are specific to the Profile Library Plug-In and are associated with every profile.

- **Name**: This property specifies a name for the profile.
- **Description**: User-defined information about this profile.

Profile Properties — Profiles

These profile properties are specific to the Profile Library Plug-In and are associated with every profile. Default values are automatically generated for each property.

- **ID**: This property is a unique identifier in the form of a GUID which links Universal Device Driver channels to profiles defined in the Profile Library Plug-In.
- **Script**: This property contains the JavaScript code that implements the required event handlers and business logic for the Universal Device Driver to communicate with a device or to operate on a file. To load a script file, select the browse ellipses (...) on the right side of the text box and choose a JavaScript source file from disk.
  Notes:
    - Script files must be UTF-8 encoded and can be created and edited offline in any editor.
    - If changes are made to the script, the file must be uploaded again.
    - If there are any channels linked to a profile, it is necessary to reinitialize the server after uploading the script.

Creating a Profile Script

The profile script is a property of the profile that contains the JavaScript to execute functions required to validate tags and communicate with a device or operate on a file. The script can be created and edited offline in any editor.

Required Functions

For Universal Device Driver to perform basic I/O operations with a device or to operate on a file, the script must contain specific functions. Some required functions are specific to Ethernet mode profile scripts while others are specific to File mode profile scripts. *See the required functions based on profile modes below.*
All scripts must define the following functions:

- onProfileLoad
- onValidateTag

onProfileLoad

The **onProfileLoad** function defines the interface contract between the profile and the driver.

Input

None

Output

onProfileLoad must return a JavaScript object with the following fields:

- version – (string) Version string, with format <Major.Minor> (for example, "2.0").

  Note: The only currently supported version is "2.0". Any other value is rejected by the driver, leading to failure of all subsequent script functions.
- mode – (string) The operation mode of the driver. Valid values are Client, Server, and File. The field is optional and defaults to Client.
  - In Client and Server mode, the Universal Device Driver requires the implementation within the script of the Ethernet mode specific functions. In Client mode, the Universal Device Driver acts as a client and opens a connection to the device. In Server mode, the Universal Device Driver acts as a server and the port is set to listen.
    See *Required Functions for Ethernet Mode Profile Script*
  - In File mode, the Universal Device Driver requires the implementation within the script of the File mode specific functions.
    See *Required Functions for File Mode Profile Script*

onValidateTag

This function is called when the address, data type, and read-only mode of a tag must be validated. The function can correct or modify a tag address, data type and read-only mode if necessary. For example, it can change the format slightly to enforce consistency among tag addresses, such as 'k01' adjusted to 'k0001'. Similarly, the onValidateTag function can correct or modify the data type and read-only tag properties. Additionally, this function can assign a bulk ID to each tag. The Universal Device Driver groups the tags with the same bulk ID together when providing tags to the onTagsRequest, onData, onFileTagsRequest, onFileOperations, or onFileChange functions.

Input

onValidateTag has a single input argument. The argument is a JavaScript object with the following fields:

- tag – (object) Represents the tag to be validated. It has the following fields:
  - address – (string) Tag address.
  - dataType – (string) server data type *(see Data Types section for valid values)*.
  - readOnly – (Boolean) true for read only, false for read/write tags.

Output

onValidateTag must return a JavaScript Object with the following fields:

- address – (string) Tag address. Can be modified if necessary, e.g. expanding 'k01' to 'k0001' (optional).
- dataType – (string) server data type, modified if necessary *(see Data Types section for valid values)* (optional).

  Note: A return value of Default is invalid; a data type must be specified if the input value is Default.
- readOnly – (Boolean) true for read only, false for read/write tags, modified if necessary (optional).

- bulkId – (number) Integer value that identifies the group into which to bulk the non-array tag with other non-array tags. If no bulkId is provided and the dataType and arrayColumns fields do not indicate an array tag, the Universal Device Driver assigns a unique value for the bulkId. The bulkId assigned is permanent and cannot be altered during this runtime session (*see Bulk Tags section for more information*) (optional).
  🌱**Tip**: If the script defines a bulkId for one non-array tag, it must define the bulkId for every non-array tag. Otherwise, the default value chosen by the Universal Device Driver may conflict with a value previously chosen by the script.

- arrayColumns – (number) Integer value that specifies the number of columns in the array. A non-zero number is required if the assigned data type is a supported array data type *(see Data Types section for valid array data type values)*. Valid values are 1 – 1000. However, if the data type is a String Array, the maximum number of columns is 50. If the data type is not an array data type, the arrayColumns must be undefined *(see Array Tags section for more information)* (optional).

  🌱**Tip**: The bulkId is ignored for tags defined with an array dataType and a non-zero arrayColumns.

- valid – (Boolean) true if the tag is valid, false if the tag is invalid (required).

Additional Script Information

The script is a collection of functions called and executed when needed. The user can add as many extra functions as desired to simplify complex operations.

🌐 **See Also**: *Profile Library Modbus Tutorial, UDD File Connectivity Guide (visit ptc.com or contact support)*

Script Writing Best Practices

- Use short meaningful variable and function names:
  - string xyz; // this variable name does not describe what it is
  - string tagAddress; // this variable is clearly used to hold the tag address value
- Comment thoroughly and remember that good code explains itself
- Create functions that do one thing
  - ConvertStringToByteArrayAndCreateMessage(){} // This function is responsible for too many tasks. This also makes the function hard to read.
- Avoid using Global variables as much as possible; use local variables instead. Global state is saved between function calls so global variables retain their value.
- Be careful of while loops. If while loops are done improperly, they can loop forever and the server can timeout and fail the operation.

🌐 **See Also**: *Script Utility Functions*

Required Functions for Ethernet Mode Profile Script

For Universal Device Driver to perform basic IO operations with a device the script must contain the following functions to handle events. The user can add additional functions, classes, variables, and global state as desired to simplify complex operations.

- onProfileLoad *(see Required Functions)*
- onValidateTag *(see Required Functions)*
- onTagsRequest
- onData

onTagsRequest

This function is called when tag values need to be read or written. It is up to the script to determine what action is taken next to execute the read or write and complete the tags. Depending on the protocol being implemented, the tags may be completed right away, additional device I/O may be required, or there may be no way to complete the tags at this time.

Input

onTagsRequest has a single input argument. The argument is a JavaScript Object with the following fields:

- type – (string) the type of operation to perform. The value is Read or Write (optional).
- tags – (Array of Objects) Array of tag objects being read or written. Each tag object represents a bulk tag or an array tag and has the following fields:
    - address – (string) tag address (optional).
    - value – (*) desired value of the tag. This field is only populated when type is Write (optional).
    - dataType – (string) server data type *(see Data Types section for valid values)* (optional).
    - readOnly – (Boolean) true for read only; false for read/write tags (optional).
    - bulkId – (number) integer value that identifies the group to which the tag belongs (optional). The bulkId is assigned in onValidateTag and cannot be altered during this runtime session *(see Bulk Tags for more information).*
    - arrayColumns – (number) integer value that specifies the number of columns in the array when the tag dataType is an array data type *(see Array Tags section for more information)* (optional).
  - ⬤ **Note**: Driver may provide one to many bulk tags or one array tag per function call.

Output
onTagsRequest must return a JavaScript object with the following fields:

- action – (string) The driver's next action. Valid return actions are Receive, Complete, or Fail (optional).
    - Receive – Indicates that the current transaction is not complete and that data is expected imminently from the device.
    - Complete – Indicates that the transaction is complete and no further I/O with the device is needed. If tags were being read, any tags returned with values are completed at this time.
    - Fail – Indicates a serious failure has occurred. All other return fields are ignored, any tags that were requested failed, and the device is in error state.
- data – (Array of numbers) Data to send to the device (optional). If data field is undefined then no data is sent. Values must be between 0 and 255 (optional).
- tags – (Array of Objects) Completed tag objects (optional). If the action is Complete, tag objects returned in this array are completed. This field is only required when type is Read. Each tag object has the following fields:
    - address – (string) Tag address
    - value – (*) New value of the tag (optional). If this field is undefined, the tag is completed with bad quality. Additionally, all tags bulked with this tag are completed with bad quality *(see Bulk Tags for more information)*. If this tag has an array data type and array columns, the array tag value must contain a value for each element *(see Array Tags for more information)*. String array elements are truncated at 1000 characters each. If a valid value is defined and the quality field is undefined, the tag is completed with good quality.
    - quality – (string) Tag quality (optional). Valid quality strings are Good, Bad, or Uncertain.
        - Good – Indicates the quality of the tag value is good. If quality is Good, then the value field must be defined. If the quality field is undefined and the value field is defined, then Good is the default quality.
        - Bad – Indicates the tag value is not useful. A tag value is not required or expected. If the value field is undefined, then Bad is the default quality.
        - Uncertain – Indicates the quality of the value of the tag is uncertain. This quality is available to allow the script writer to indicate an issue with the validity, staleness, or out of range state of the value being provided. The value field must also be defined.

onData
This function is called whenever the Universal Device Driver receives data from the device. If there are any uncompleted tags when onData is called, the tags and type (Read or Write) are included along with the data received. If the Universal Device Driver receives data and there are no uncompleted tags, then only the data input field is populated.
⬤ **See Also**: *Script Utility Functions*

Input
onData has a single input argument. The argument is a JavaScript Object with the following fields:

- data – (Array of numbers) Data that was received by the Universal Device Driver (optional).
- type – (string) Type of operation. The value is Read or Write. This field is undefined if there are no uncompleted tags (optional).
- tags – (Array of Objects) Array of tag objects being read or written (optional). This field is undefined if there are no uncompleted tags. Each tag object represents a bulk tag or an array tag and has the following fields:
    - address – (string) Tag address
    - value – (*) Desired value of the tag. This field only exists when type is Write.
    - dataType – (string) server data type *(see Data Types section for valid values)*.
    - readOnly – (Boolean) true for read only; false for read/write tags.
    - bulkId – (number) integer value that identifies the group to which the tag belongs (optional). The bulkId is assigned in onValidateTag and cannot be altered during this runtime session *(see Bulk Tags section for more information)*.
    - arrayColumns – (number) integer value that specifies the number of columns in the array when the tag dataType is an array data type *(see Array Tags section for more information)* (optional).
    - **Note**: Driver may provide one to many bulk tags or one array tag per function call.

Output
onData must return a JavaScript object with the following fields:

- action – (string) The driver's next action. Valid return actions are Receive, Complete, or Fail (optional).
    - Receive – Indicates that the current transaction is not complete and that more data is expected imminently from the device.
    - Complete – Indicates that the transaction is complete and no further I/O with the device is needed. If tags were being read, then any tags returned with values are completed at this time.
    - Fail – Indicates a serious failure has occurred. All other return fields are ignored, any uncompleted tags fail, and the device is in error state.
- data – (Array of numbers) Data to send to the device (optional). If data field is undefined, no data is sent. Values must be between 0 and 255.
- tags – (Array of Objects) Completed tag objects (optional). If the action is Complete, tag objects returned in this array are completed. This field is only required when type is Read. Each tag object element has the following fields:
    - address – (string) Tag address.
    - value – (*) New value of the tag (optional). If this field is undefined then the tag is completed with bad quality. Additionally, all tags bulked with this tag are completed with bad quality *(see Bulk Tags for more information)*. If this tag has an array data type and array columns, the array tag value must contain a value for each element *(see Array Tags for more information)*. String array elements are truncated at 1000 characters each. If a valid value is defined and the quality field is undefined, the tag is completed with good quality.
    - quality – (string) Tag quality (optional). Valid quality strings are Good, Bad, or Uncertain.
        - Good – Indicates the quality of the tag value is good. If quality is Good, then the value field must be defined. If the quality field is undefined and the value field is defined, then Good is the default quality.
        - Bad – Indicates the tag value is not useful. A tag value is not required or expected. If the value field is undefined, then Bad is the default quality.
        - Uncertain – Indicates the quality of the value of the tag is uncertain. This quality is available to allow the script writer to indicate an issue with the validity, staleness, or out of range state of the value being provided. The value field must also be defined.

See Also: Script Utility Functions, Logging Functions, Bulk Tags

Required Functions for File Mode Profile Script

For Universal Device Driver to perform file operations the script must contain the following functions to handle events. The user can add additional functions, classes, variables, and global state as desired to simplify complex operations.

- onProfileLoad *(see Required Functions)*
- onValidateTag *(see Required Functions)*

- onFileTagsRequest
- onFileOperations
- onFileChange

onFileTagsRequest

This function is called when tag values need to be read or written. It is up to the script to determine what action is taken next to complete the tags. The tags may be completed right away, or file operations may be requested and the tags completed later.

Input

onFileTagsRequest has a single input argument. The argument is a JavaScript Object with the following fields:

- type – (string) the type of operation to perform. The value is Read or Write (optional).
- tags – (Array of Objects) Array of tag objects being read or written. Each tag object represents a bulk tag or an array tag and has the following fields:
    - address – (string) tag address (optional).
    - value – (*) desired value of the tag. This field is only populated when type is Write (optional).
    - dataType – (string) server data type *(see Data Types for valid values)* (optional).
    - readOnly – (Boolean) true for read only; false for read/write tags (optional).
    - bulkId – (number) integer value that identifies the group to which the tag belongs (optional). The bulkId is assigned in onValidateTag and cannot be altered during this runtime session *(see Bulk Tags for more information)*.
    - arrayColumns – (number) integer value that specifies the number of columns in the array when the tag dataType is an array data type *(see Array Tags section for more information)* (optional).

        ● **Note**: Driver may provide one to many bulk tags or one array tag per function call.

Output

onFileTagsRequest must return a JavaScript object with the following fields:

- action – (string) The driver's next action. Valid return actions are Operate, Complete, or Fail (optional).
    - Operate - Indicates that the current transaction is not complete and that a file operation is requested.
    - Complete - Indicates that the transaction is complete, and no further file operation is needed. If tags were being read, any tags returned with values are completed at this time.
    - Fail - Indicates a serious failure has occurred. All other return fields are ignored, any tags that were requested failed, and the device is in error state.
- operations (Array of strings) – The operations field contains a specific file operation to perform (optional). Required when action is Operate. Valid operations are OpenFile, CloseFile, ReadFile, WriteFile, ReadLine, WriteLine, AsyncWatchDir, AsyncUnwatchDir, CreateFile, MoveFile, and DeleteFile. Only one operation is supported (i.e. one array element). *See File Operations*
- fileOrPath (string) – The relative path of the file or directory on which to operate (optional). If the Operate request requires a file name path, this field may contain that string. However, if not defined, and the Operate request requires a file name path, then the driver uses the value entered in the Universal Device Driver File Name Device property. If the Operate request requires a path, such as AsyncWatchDir, this field may contain that string. However, if not defined, and the Operate request requires a path, then the driver uses the value entered in the Universal Device Driver Base Path device property. *See the* Universal Device Driver *help file Device properties .*
- data (Array of numbers) – The file operation may require data (optional). *See File Operations for definition of data required for each operation.*
- tags – (Array of Objects) Completed tag objects (optional). If the action is Complete, tag objects returned in this array are completed. This field is only required when type is Read. Each tag objects element has the following fields:
    - address – (string) Tag address
    - value – (*) New value of the tag (optional). If this field is undefined, the tag is completed with bad quality. Additionally, all tags bulked with this tag are completed with bad quality (see Bulk Tags for more information). If this tag has an array data type and array columns, the array tag value must contain a value for each element *(see Array Tags for more information)*. String array

elements are truncated at 1000 characters each. If a valid is defined and the quality field is undefined, the tag is completed with good quality.

- ○ quality – (string) Tag quality (optional). Valid quality strings are Good, Bad, or Uncertain.
  - Good – Indicates the quality of the tag value is good. If quality is Good, then the value field must be defined. If the quality field is undefined and the value field is defined, then Good is the default quality.
  - Bad – Indicates the tag value is not useful. A tag value is not required or expected. If the value field is undefined, then Bad is the default quality.
  - Uncertain – Indicates the quality of the value of the tag is uncertain. This quality is available to allow the script writer to indicate an issue with the validity, staleness, or out of range state of the value being provided. The value field must also be defined.

onFileOperations

This function is called whenever the Universal Device Driver completes an Operate action previously requested from the script. If there are any uncompleted tags when onFileOperations is called, the tags and message type (Read or Write) are included along with the file operation performed, the file operation result, and the file name path or directory operated on.

Input

onFileOperations has a single input argument. The argument is a JavaScript Object with the following fields:

- type – (string) the type of operation to perform. The value is Read or Write (optional).
- tags – (Array of Objects) Array of tag objects being read or written. Each tag object represents a bulk tag or an array tag and has the following fields:
  - ○ address – (string) tag address (optional).
  - ○ value – (*) desired value of the tag. This field is only populated when type is Write (optional).
  - ○ dataType – (string) server data type *(see Data Types for valid values)* (optional).
  - ○ readOnly – (Boolean) true for read only; false for read/write tags (optional).
  - ○ bulkId – (number) integer value that identifies the group to which the tag belongs (optional). The bulkId is assigned in onValidateTag and cannot be altered during this runtime session *(see Bulk Tags for more information)*.
  - ○ arrayColumns – (number) integer value that specifies the number of columns in the array when the tag dataType is an array data type *(see Array Tags section for more information)* (optional).
  - Note: Driver may provide one to many bulk tags or one array tag per function call.
- lastOperation (string) – The last file operation performed. Possible values are OpenFile, CloseFile, ReadFile, WriteFile, ReadLine, WriteLine, AsyncWatchDir, AsyncUnwatchDir, CreateFile, MoveFIle, and DeleteFile. *See File Operations section*.
- lastOperationResult (string) – The result of the last file operation.
- lastFileOrPath (string) – The relative path of the last file or path operated on during the operation.

Output

onFileOperations must return a JavaScript object with the following fields:

- action – (string) This driver's next action. Valid return actions are Operate, Complete, or Fail (optional).
  - ○ Operate – Indicates that the current transaction is not complete and that a file operation is requested.
  - ○ Complete – Indicates that the transaction is complete, and no further file operation is needed. If tags were being read, any tags returned with values are completed at this time.
  - ○ Fail – Indicates a serious failure has occurred. All other return fields are ignored, any tags that were requested failed, and the device is in error state.
- operations (Array of strings) – The operations field contains a specific file operation to perform (optional). Required when action is Operate. Valid operations are OpenFile, CloseFile, ReadFile, WriteFile, ReadLine, WriteLine, AsyncWatchDir, AsyncUnwatchDir, CreateFile, MoveFile, and DeleteFile. Only one operation is supported (i.e. one array element). *See File Operations*.
- fileOrPath (string) – The relative path of the file or path on which to operate (optional). If the Operate request requires a file name path, this field may contain that string. However, if not defined, and the

Operate request requires a file name path, then the driver uses the value entered in the File Name Device property. If the Operate request requires a file path, such as AsyncWatchDir, this field may contain that string. However, if not defined, and the Operate request requires a path, the driver uses the value entered in the Base Path device property.

- data (Array of numbers) – The file operation may require data (optional). *See File Operations section for definition of possible data required for each operation.*
- tags – (Array of Objects) Completed tag objects (optional). If the action is Complete, tag objects returned in this array are completed. This field is only required when type is Read. Each tag object has the following fields:
    - address - (string) Tag address
    - value - (*) New value of the tag (optional). If this field is undefined, the tag is completed with bad quality. Additionally, all tags bulked with this tag are completed with bad quality *(see Bulk Tags for more information)*. If this tag has an array data type and array columns, the array tag value must contain a value for each element *(see Array Tags for more information)*. String array elements are truncated at 1000 characters each. If a value is defined and the quality field is undefined, the tag is completed with good quality.
    - quality - (string) Tag quality (optional). Valid quality strings are Good, Bad, or Uncertain.
        - Good - Indicates the quality of the tag value is good. If quality is Good, then the value field must be defined. If the quality field is undefined and the value field is defined, then Good is the default quality.
        - Bad - Indicates the tag value is not useful. A tag value is not required or expected. If the value field is undefined, then Bad is the default quality.
        - Uncertain - Indicates the quality of the value of the tag is uncertain. This quality is available to allow the script writer to indicate an issue with the validity, staleness, or out of range state of the value being provided. The value field must also be defined.

onFileChange

This function is called whenever the Universal Device Driver needs the script to process a tag read or write but has detected a change in the directory previously requested to be watched with an AsyncWatchDir file operation request. The tags and message type (Read or Write) are included along with the change detected, and the file name path or directory impacted by the change.

Input

onFileChange has a single input argument. The argument is a JavaScript Object with the following fields:

- type – (string) the type of operation to perform. The value is Read or Write (optional).
- tags – (array of objects) array of tag objects being read or written. Each tag object represents a bulk tag or an array tag and has the following fields:
    - address - (string) tag address (optional).
    - value - (*) desired value of the tag. This field is only populated when type is Write (optional).
    - dataType - (string) server data type *(see Data Types for valid values) (optional)*.
    - readOnly - (Boolean) true for read only; false for read/write tags (optional).
    - bulkId - (number) integer value that identifies the group to which the tag belongs (optional). The bulkId is assigned in onValidateTag and cannot be altered during this runtime session *(see Bulk Tags for more information)*.
    - arrayColumns - (number) integer value that specifies the number of columns in the array when the tag dataType is an array data type *(see Array Tags section for more information)* (optional).

    Note: Driver may provide one to many bulk tags or one array tag per function call.

- change (string) - The detected change. Possible values are Created, Deleted, RenamedFrom, RenamedTo, Modified, or Invalid.
- fileOrPath (string) - The relative path of the file or directory that changed.

Output

onFileChange must return a JavaScript object with the following fields:

- action - (string) This driver's next action. Valid return actions are Operate, Complete, or Fail (optional).
    - Operate - Indicates that the current transaction is not complete and that a file operation is requested.
    - Complete - Indicates that the transaction is complete, and no further file operation is needed. If tags were being read, any tags returned with values are completed at this time.
    - Fail - Indicates a serious failure has occurred. All other return fields are ignored, any tags that were requested failed, and the device is in error state.
- operations (array of strings) - The operations field contains a specific file operation to perform (optional). Required when action is Operate. Valid operations are OpenFile, CloseFile, ReadFile, WriteFile, ReadLine, WriteLine, AsyncWatchDir, AsyncUnwatchDir, CreateFile, MoveFile, and DeleteFile. Only one operation is supported (i.e., one array element). *See File Operations*
- fileOrPath (string) - The relative path of the file or path on which to operate (optional). If the Operate request requires a file name path, this field may contain that string. However, if not defined, and the Operate request requires a file name path, then the driver uses the value entered in the File Name Device property. If the Operate request requires a file path, such as AsyncWatchDir, this field may contain that string. However; if not defined, and the Operate request requires a path, the driver uses the value entered in the Base Path device property.
- data (array of numbers) - The file operation may require data (optional). *See File Operations for definitions of data required for each operation.*
- tags - (array of objects) Completed tag objects (optional). If the action is complete, tag objects returned in this array are completed. This field is only required when type is read. Each tag object element has the following fields:
    - address - (string) tag address.
    - value - (*) new value of the tag (optional). If this field is undefined, the tag is completed with bad quality. Additionally, all tags bulked with this tag are completed with bad quality *(see Bulk Tags for more information)*. If this tag has an array data type and array columns, the array tag value must contain a value for each element *(see Array Tags for more information)*. String array elements are truncated at 1000 characters each. If a valid value is defined and the quality field is undefined, the tag is completed with good quality.
    - quality - (string) tag quality (optional). Valid quality strings are good, bad, or uncertain.
        - Good - indicates the quality of the tag value is good. If quality is Good, then the value field must be defined. If the quality field is undefined and the value field is defined, then good is the default quality.
        - Bad - indicates the tag value is not useful. A tag value is not required or expected. If the value field is undefined, then Bad is the default quality.
        - Uncertain - indicates the quality of the value of the tag is uncertain. This quality is available to allow the script writer to indicate an issue with the validity, staleness, or out of range state of the value being provided. The value field must also be defined.

🌢 *See Also: Script Utility Functions, Logging Functions, Bulk Tags, Array Tags, File Operations*

File Operations

When the script provides the File profile mode in onProfileLoad to the Universal Device Driver any of the required File mode API functions can request various file operations. The fields provided by the script Operate request depend on the operation.

- **OpenFile** - Opens a file with option flags. File must be opened before operations such as ReadFile, WriteFile, ReadLine, WriteLine, or CloseFile can be requested.
    - fileOrPath: <file> - The Universal Device Driver opens the file at this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.
    🔶 **Caution**: This file path may not contain any of the characters: < > : " / | ? * or end with a period, and is limited to a maximum of 244. This file path is added to the end of the configured Base Path and this fully-qualified file name is limited to 258 characters.
    - data: [<mode>, <type>, <access>, <createmode>] - The open flags that specify how the file should be opened. All four bytes must be provided.

- - <mode> - File access mode. Set to one of the following byte values: 0 - Read only (default); 1 - Write only; 2 - Read and write access.
    - <type> - Character mode. Set to one of the following byte values: 0 - Binary mode (default); 1 - Text mode with special processing for carriage return - line feed pairs.
    - <access> - Sharing mode. Set to one of the following byte values: 0 - Deny read and write access to others (default); 1 - Deny write access to others; 2 - Deny read access to others; 3 - No sharing restrictions.
    - <createmode> - File creation mode. Set to one of the following byte values: 0 - Open existing (default); 1 - Create a new file if no file exists, otherwise truncate existing; 2 - Create a new file if no file exists, otherwise opens existing.
- **CloseFile** - Closes a file. File must have been opened with a previous file operation OpenFile request. The file name provided to this file operation must match the file name provided to the OpenFile request.
  - fileOrPath: <file> - The Universal Device Driver closes the file at this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.
- **ReadFile** - Read entire contents of a file. File must have been opened with a previous file operation OpenFile request. The file name provided to this file operation must match the file name provided to the OpenFile request. File must be opened in Binary mode, even if data being read is text. The file contents are returned to the script in the data field in the call to the onFileOperations function upon ReadFile completion.
  - fileOrPath: <file> - The Universal Device Driver reads the entire file at this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.
- **WriteFile** - Write entire contents to a file. File must have been opened with a previous file operation OpenFile request. The file name provided to this file operation must match the file name provided to the OpenFile request. File must be opened in Binary mode, even if data being written is text. The number of bytes written is returned to the script in the four bytes of the data field in the call to the onFileOperations function upon WriteFile completion. Byte order: data[0] - lo byte lo word; data[1] - hi byte lo word; data[2] - lo byte hi word; data[3] - hi byte hi word.
  - fileOrPath: <file> - The Universal Device Driver writes the entire contents to the file at this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.
  - data: <data> - Binary or text data bytes to write to the file (required).
- **ReadLine** - Read a single line of text from a file, from the current position or from the beginning of the file. File must have been opened with a previous file operation OpenFile request. The file name provided to this file operation must match the file name provided to the OpenFile request. File must be opened in Text mode and only ANSI files are supported. Files without Windows line endings (CRLF) may not read properly using this operation. The bytes read are returned to the script in the data field in the call to the onFileOperations function upon ReadLine completion. The script must convert this byte data back to text.
  - fileOrPath: <file> - The Universal Device Driver reads a line from this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.
  - data: [<position>] (optional)
    - <position> - File position to read from. Valid values are 0 - Read from current position, no seek (default); 1 - Read from the beginning of the file, seek to beginning; 2 - Read from the end of the file, seek to end. Note that reading from the end of the file results in an end-of-file error.
- **WriteLine** - Write a single line of text to a file. File must have been opened with a previous file operation OpenFile request. The file name provided to this file operation must match the file name provided to the OpenFile request. File must be opened in Text mode and only ANSI files are supported. The number of bytes written is returned to the script in the four bytes of the data field in the call to the onFileOperations function upon WriteLine completion. Byte order: data[0] - lo byte lo word; data[1] - hi byte lo word; data[2] - lo byte hi word; data[3] - hi byte hi word.

- ○ fileOrPath: <file> - The Universal Device Driver writes the contents of a single line to the file at this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.
        - ○ data: [<position>, <text data bytes>] (required)
            - • <position> - File position to write to. Valid position values are 0 – Write to current position, no seek (not supported, reserved for future use); 1 – Write to the beginning of the file, seek to beginning(not supported, reserved for future use); 2 – Write to the end of the file, seek to end (default).
            - • <text data bytes> - Specify the text data bytes to write. Must contain at least one byte.
- • **AsyncWatchDir** – Start asynchronous monitoring of a directory. Directory changes are reported through the onFileChanges function. Changes include create, delete, rename, and modification of a file or subdirectory. Only one directory can be monitored at a time.
    - ○ fileOrPath: <path> - The Universal Device Driver monitors this relative path. The driver adds <path> to the end of the base path which is configured in its Base Path Device property. (optional) If <path> is undefined, the Universal Device Driver uses the Base Path Device property value.
- • **AsyncUnwatchDir** – Stop asynchronous monitoring of a directory. A previous file operation request to AsyncWatchDir must have occurred in order to stop watching.
- • **CreateFile** – Creates a file.
    - ○ fileOrPath: <file> - The Universal Device Driver creates the file at this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.
- • **MoveFile** – Moves a non-open, existing file to specified destination file path.
    - ○ fileOrPath: <sourcefile> - The Universal Device Driver moves the file at this relative file path. The driver adds <sourcefile> to the end of the base path which is configured in its Base Path Device property. This <sourcefile> may contain subdirectories but must be relative to the Base Path. (optional) If <sourcefile> is undefined, the Universal Device Driver uses the File Name Device property value. <sourcefile> cannot be open by the script or any other application and it must exist.
    - ○ data: <destinationfile> - Specify the text bytes representing the destination file path. The driver adds <destinationfile> to the end of the base path which is configured in its Base Path Device property. This <destinationfile> may contain subdirectories but must be relative to the Base Path. (required) Must contain at least one byte. The destination file path must not exist.
- • **DeleteFile** – Deletes a file.
    - ○ fileOrPath: <file> - The Universal Device Driver deletes the file at this relative file path. The driver adds <file> to the end of the base path which is configured in its Base Path Device property. This <file> may contain subdirectories but must be relative to the Base Path. (optional) If <file> is undefined, the Universal Device Driver uses the File Name Device property value.

Data Types

The dataType strings used by the profile are derived from the data types supported by the server. The valid values for dataType are:

- • Default
  Note: This is not a valid return value; Defaults only appear as an input.
- • String
- • String Array
- • Boolean
- • Boolean Array
- • Char
- • Char Array
- • Byte
- • Byte Array

- Short
- Short Array
- Word
- Word Array
- Long
- Long Array
- DWord
- DWord Array
- Float
- Float Array
- Double
- Double Array
- BCD
- LBCD
- LLong
- LLong Array
- QWord
- QWord Array

Bulk Tags

To reduce requests to the device, improve throughput from a device, process tags faster, or obtain similar data at the same time, the script can define the bulkId field in the result of the onValidateTag function for each non-array tag. Tags that share the same bulkId are blocked together and provided to the onTagsRequest, onFileTagsRequest, onData, onFileOperations, and onFileChange functions allowing the script to complete all the tags in the block.

If script defines one bulkId, it must define a bulkId for all non-array tags

If the bulkId is not defined in the onValidateTag function, the Universal Device Driver assigns a unique value to each tag that does not have the arrayColumns defined.
Caution: If the script defines the bulkId of one non-array tag, it must define a bulkId for every non-array tag. Otherwise, the value chosen by the Universal Device Driver may conflict with a value previously chosen by the script.

Tags with a valid array data type and non-zero arrayColumns are not blocked

The Universal Device Driver does not allow blocking array tags. If a bulkId is assigned to a tag in onValidateTag with a valid array data type and arrayColumns, the bulkId is ignored. The bulkId of the array tag is 0 when received in other API functions.

Block Size Performance vs. CPU Usage

The onValidateTag function does not limit the number of tags defined with the same bulkId. However, to prevent unnecessary strain on the script engine, the Universal Device Driver fails Read requests if more than 8000 tags are in the same block.
Caution: The script writer should monitor the performance and CPU usage of the server runtime and script engine processes to ensure that the size of the block is optimal for the use case.

Example with Modbus Protocol

Modbus devices allow requests to specify a certain number of registers starting with a particular address. The script could assign the same bulkId to those tags that could be included in the same request to the device. This reduces the number of transactions required over the wire and improves performance.

Example with MQTT Pub / Sub Protocol

The script could assign the same bulkId to all the tags in the same topic to allow updating them at the same time. This would allow the entire transaction to be processed together, improving performance.

Example with Structure

The tags in a structure often represent the state of the device cycle at a given moment in time. If the tags are updated one at a time, there is no guarantee that the data in one tag was related to the data in another tag. Assigning the same bulkId to those related tags allows the script to issue the necessary requests to the device, save the results, and ultimately update the tags at the same time, ensuring the data is related. For example, if a tag holds the temperature from a temperature sensor and another tag holds the time that was taken, in a non-bulked configuration, the temperature tag could be from a previous read from the time tag. This is dependent on how many tags are in the project and how quickly they are scanned. But if only processing one tag at a time, there is no guarantee of the relationship of the data to other tag data. However, if the tags were assigned the same bulkId, then the script can control the update of the data and ensure that relationship.

Array Tags

The script can define a tag as an array tag by assigning the dataType and arrayColumns fields in the onValidateTag function. Tag validation fails if:

- The dataType is an array data type and the arrayColumns is undefined or zero.
- The dataType is an array data type and the arrayColumns exceeds the allowed value for the data type. The maximum array column value is 1000 for most array data types. However, the maximum array column value for string array is 50.
- The dataType is not an array data type and the arrayColumns is defined (zero or non-zero).

Tags with a valid array data type and non-zero arrayColumns are not blocked

The Universal Device Driver does not allow blocking array tags. If a bulkId is assigned to a tag in onValidateTag with a valid array data type and arrayColumns, the bulkId is ignored. The bulkId of the array tag is 0 when received in other API functions.

Element data must be coercible to the assigned data type of the tag

The array data provided in the tag value field when tags are completed in onTagsRequest, onFileTagsRequest, onData, onFileOperations, and onFileChange is type-less and can therefore have mixed types. The Universal Device Driver must derive the "best fit" type from the values in the array. Different numeric types can co-exist. To accommodate these various types, the driver processes the array elements and determine the type that best fits the current element as well as all the previous elements in the array. If the value of any element cannot be coerced to the data type of the array tag, a "Script result invalid" failure is returned.

🟢 **Tip**: Types are converted up as follows: unsigned 64-bit integer (QWord) -> signed 64-bit integer (LLong) -> 64-bit real value (Double).

Example:

Given an array value of [0,-1,2.3]. The Universal Device Driver considers the array a QWord when it encounters the value 0, converts to LLong when it encounters the -1, and converts again to Double when it encounters 2.3. Therefore [0,-1,2.3] is treated as a an array of Doubles.

⚪ **Note**: 1.0, 2.0, etc., are treated as an UInt64 even though it is specified as a Double.

⚪ **Note**: String array element values are truncated at 1000 characters.

Profile Inputs

All properties in the Profile Inputs device property group are passed to the script onProfileInputsChange function when they are changed by the user. If a property in the Profile Inputs group is changed, and onProfileInputsChange is not correctly implemented, a script engine error occurs.

onProfileInputsChange

This function is called whenever the Universal Device Driver needs the script to process a tag read or write, but has detected a change in any of the Profile Inputs properties. The function is optional and is not called if the Profile Inputs properties remain at their default empty values. It is called immediately before onTagsRequest in Ethernet mode or onFileTagsRequest in File mode. The function must return the result of the Profile Inputs properties validation.

🟢 It is recommended that onProfileInputsChange saves whatever data is needed from the property to a variable or constant that can be read by other parts of the profile. *See the example below.*

Input

onProfileInputsChange has a single input argument. The argument is a JavaScript Object with the following field:

- stringInput – (string) The latest value of the Input String property.

🌱 It is converted to a UTF-8 string before being passed to the profile and it is truncated to 20 kb.

Output
onProfileInputsChange must return a JavaScript object with the following field:

- valid – (boolean) Indicates if the profile considers the input value(s) valid. If it is false, a warning is posted to the Event Log.

🌱 It is recommended that the profile logs a message to inform the user of the reason for the validation failure(s), though this is not required.

Example with HTTP
HTTP requests require a port to be issued. The port can be hard coded as part of the profile; however, it can be useful to re-use the same profile on multiple channels to communicate with multiple ports. onProfileInputsChange can be used to interpret the input string as the port to be used in the HTTP request. If there are additional unique requirements, like an API key, the input string can be JSON formatted, which can be parsed into separate key/-value pairs with JSON.parse.

Provided Functions

To aid the script writer in writing the profile scripts, the script engine environment (Script Engine SDK) provides the following helper functions and classes:

- Log: allows users to log messages to the server event log
- Cache utility: a cache implementation that stores tag data

Logging Functions

The log() function can be used to log messages to the server event log. When an error is encountered, it is considered best practice to log a message with helpful information about what happened and return a status of failure rather than throwing an exception.

Utility Functions

In addition to the required functions, a user can create as many additional functions as desired to simplify complex operations. The sample profiles provide examples of helper functions that can assist with converting data types or determining the validity of the data returned from a device.

Caching Functions
For unsolicited communications it is necessary to manage a cache so OPC clients can retrieve and update tag data. To help facilitate this there are three caching functions designed to be used in the script: initalizeCache, writeToCache, and readFromCache.

initalizeCache
Initializes the cache. Calling this function allows users to set the maximum cache size.

Input
initalizeCache has one input argument:

- maxSize – (number) the maximum size of the cache (10,000 maximum)

Output
None

writeToCache
Inserts or overwrites a key-value pair in the cache.

Input
writeToCache has two input arguments:

- key - (string) The tag address
- value - (*) The value of the tag. Maximum length of 4096 characters

Output

writeToCache returns a string indicating if the cache was updated successfully. Possible return values are success and error.
Possible causes of an error include the following:

- Any of the arguments passed to the function are undefined or of the wrong type
- The cache size limit has been reached
- The value argument has exceeded its maximum length limit of 4096 characters

readFromCache

Retrieves a key-value pair stored in the cache. If the key-vale pair is not found, the return object's value property has a value of undefined.

Input

readFromCache has one input argument:

- key - (string) The tag address to retrieve data from

Output

readFromCache returns an object that contains the following properties:

- key - (string) The tag address
- value - (*) The last known value of the tag. Undefined if the key is not found in the cache. See below for possible reasons this could occur.

Reasons the value property could be undefined:

- If a key-value pair has not been entered into the cache at the time the readFromCache is called.
- If the cache is full and a key-value pair needs to be added, the cache ages out the first key-value pair updated more than 24 hours prior. Because of this, it is possible that the key being read from the cache is aged out and no longer exists in the cache.

Using the Configuration API

This section describes the process to create a profile using the Configuration API Service. The steps shown here can be used to create any of the profile types. For a template of functions required for a specific profile type, create a new profile with no script defined in the request body, then send a GET API request.

🌼 **Tip**: This documentation assumes the user is on the same machine as the server and is using the default HTTP port. Therefore, localhost:57412 is used as the address for all API calls. Change the IP address and port as needed.

Sending API Requests

The API is accessible through a REST interface that can act on HTTP requests.

🔵 *See the Configuration API Service help documentation in server help under Configuration API Service section for more information about interfacing with the server over the API.*

Creating a Profile

To create a profile using the API, send a POST to the following endpoint:

```
POST http://localhost:57412/config/v1/project/_profile_library/profiles
```

with a body:

```
{
"common.ALLTYPES_NAME": "Profile_Name_Here"
}
```

If the profile is created with only its name defined in the POST request, the server populates the Script and ProfileID fields. The Script field then contains a template script that can be retrieved with a GET request (see View an existing Profile) as a starting point.

The user can optionally specify a description; the JavaScript that makes up the "driver logic" and a reference ID in the form of a UUID. The body of a POST including these properties should look like:

```
{
"common.ALLTYPES_NAME": "Profile_Name_Here"
"common.ALLTYPES_DESCRIPTION": "description_here",
"libudcommon.LIBUDCOMMON_PROFILE_JAVASCRIPT": "<javascript>",
"libudcommon.LIBUDCOMMON_PROFILE_ID": "<UUID>"
}
```

Updating a Profile

To update a profile, send a PUT request to the endpoint, and append "/profiles/" and the profile name, in the form of:

```
PUT http://localhost:57412/config/v1/project/_profile_library/profiles/<profile_name>
```

It is not recommended to update profiles with an active client reference. When a linked profile is updated, tags on any linked channels report "bad quality" until the new script or profile configuration is propagated to each of the linked channels. At that point, assuming that the profile is valid and works correctly with the existing channel configurations, those tags restart communication and begin reporting "good" quality data again.

Updating a profile can cause linked channels to become invalid. For example, if the onValidateTag function changes and the static tag or dynamic client tag addresses no longer fit the address schema in the new function; those tags that no longer pass validation remain in "bad" quality until the profile and or link is updated or modified again.

🌼 **Tip**: Once a profile is updated, reinitialize to apply the changes.

View an Existing Profile

To view the contents in an existing profile, send a GET request to the endpoint and append "/profiles/" and the profile name, in the form of:

```
GET http://localhost:57412/config/v1/project/_profile_library/profiles/<profile_name>
```

Note: If some of the properties of the profile were generated by the server (properties that were omitted from the POST request to create the profile), they can be viewed in the GET response.

Index

A

B

C

D

U

V

W