

TwinCAT ADS Made Easy

TAME 4.1

JavaScript Library

API Reference

TAME 4.1: API Reference

Copyright © Thomas Schmidt <t.schmidt.p1 at freenet.de>, 2017
Last changed: 12.07.2017

This manual is provided „as it is“, without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Beckhoff® and TwinCAT® are registered trademarks of Beckhoff Automation GmbH.

Chapters

1. Introduction.....	1
2. Instancing the Web Service Client.....	2
Parameters (Type, Required, Default).....	4
Instance Properties (Type, Default).....	6
3. ReadReq and WriteReq.....	8
3.1 Read Request.....	8
Supported Data Types.....	10
Parameters (Type, Required, Default).....	11
Item Parameters (Type, Required, Default).....	12
3.2 Write Request.....	13
Supported Data Types.....	14
Parameters (Type, Required, Default).....	15
Item Parameters (Type, Required, Default).....	16
4. Sum Commands.....	17
4.2 Sum Read Request.....	17
Supported Data Types.....	19
Parameters (Type, Required, Default).....	20
Common Item Parameters (Type, Required, Default).....	21
Type Dependent Item Parameters (Type, Required, Default).....	22
4.2 Sum Write Request.....	23
Supported Data Types.....	24
Parameters (Type, Required, Default).....	25
Common Item Parameters (Type, Required, Default).....	26
Type Dependent Item Parameters (Type, Required, Default).....	26
5. Type Dependent Requests.....	27
5.1 Read Requests.....	27
Supported Data Types.....	29
Common Parameters (Type, Required, Default).....	30
Type Dependent Parameters (Type, Required, Default).....	31
5.2 Write Requests.....	32
Supported Data Types.....	33
Common Parameters (Type, Required, Default).....	34

Type Dependent Parameters (Type, Required, Default).....	35
6. Arrays.....	36
6.1 Read Requests.....	36
Supported Data Types.....	37
Common Parameters (Type, Required, Default).....	39
Type Dependent Parameters (Type, Required, Default).....	40
6.2 Write Requests.....	41
Supported Data Types.....	42
Common Parameters (Type, Required, Default).....	43
Type Dependent Parameters (Type, Required, Default).....	44
7. Structures.....	45
7.1 Read Request.....	46
Supported Data Types.....	49
Parameters (Type, Required, Default).....	50
7.2 Write Request.....	52
Supported Data Types.....	53
Parameters (Type, Required, Default).....	54
8. Arrays Of Structures.....	56
8.1 Read Request.....	56
Supported Data Types.....	57
Parameters (Type, Required, Default).....	58
8.2 Write Request.....	60
Supported Data Types.....	61
Parameters (Type, Required, Default).....	62
9. Handles.....	64
9.1 Getting Handles.....	65
Parameters (Type, Required, Default).....	66
9.2 Using Handles.....	67
9.3 Releasing Handles.....	68
Parameters (Type, Required, Default).....	69
9.4 Logging Handles.....	70
10. Other Methods.....	71
10.1 Import/Export the Symbol Table.....	71
Methods for Handling the Symbol Table (Type, Arguments, Return).....	71

10.2 Import/Export the Data Type Table.....	71
Methods for Handling the Data Type Table (Type, Arguments, Return).....	71
10.3 Check the PLC State.....	72
Methods for Reading the ADS State (Type, Arguments, Return).....	72
Parameters (Type, Required, Default).....	72
11. Type Dependent Parameters.....	73
11.1 REAL and LREAL.....	73
11.2 STRING.....	74
11.3 TIME.....	76
Formatting Instructions.....	78
11.4 TOD, DT and DATE.....	79
Formatting Instructions.....	81
12. Debugging.....	82
13. Tips & Tricks.....	83

1. Introduction

TAME is JavaScript library created for an easy and comfortable access to the TwinCAT ADS WebService. The name is an acronym for „TwinCAT ADS Made Easy“ and stands also for „taming“ the complexity of ADS and AJAX requests. Originally a „wast product“ from the programming of a browser based visualisation for my home, it has become a (in my opinion) useful little piece of software and I hope it will help others who want to develop their own visualisations. It allows to exchange data with a TwinCAT PLC without any knowledge of ADS. The communication is based on AJAX requests, the browser connects to the webserver running on the PLC device. If you want to send or read data through a firewall you need to forward only one port (80 by default) to connect to the WebService.

There are methods for read and write access to single variables, variable blocks, arrays and structures in the TwinCat PLC. Supported data types are BOOL, BYTE, WORD, DWORD, USINT, SINT, UINT, INT, UDINT, DINT, TIME, TOD, DT, DATE, REAL, LREAL and STRING. There is also a special „type“ named INT1DP: It's an INT in the PLC, but in JavaScript the variable is of type float with 1 decimal place (i.e. a value of 568 in the PLC is 56.8 in JavaScript).

The library provides built-in conversion of date and time values to formatted strings and REAL values can be rounded to a desired number of decimal places. Another feature is the automatic structure padding for exchanging data with TwinCAT 2 and ARM-based devices (i.e. CX90xx, 4-byte alignment) or with TwinCAT 3 (8-byte alignment). If the client parameter „alignment“ is set to the correct value you don't need to worry about the alignment of data structures.

It's possible to access single elements of structures and arrays and also internal variables and parameters of FB's. This can be done by using handles or the TPY-file generated by TwinCAT. Target information like AMS NetId and AMS Port can be read from the file, it's no longer necessary to set them manually. The biggest drawback is that the file has to be copied to the webserver each time the PLC program has changed. See chapter 9 for additional information.

This documentation is primarily intended for programmers, who are familiar with the basic concepts of the JavaScript language. I presume that the ADS WebService is already installed on the device you want to connect to. If not, visit <http://infosys.beckhoff.com> for more information about the installation of the server.

The library is Open Source licensed under the MIT and the GPLv3 license. Have fun!

IMPORTANT! Be aware that if you change the PLC program the addresses of PLC variables may also change if you don't use fixed addresses (i.e. %MX). Before you make changes to your PLC program you should close all JavaScript clients built with this library. After your changed program is online and running you can start your clients again and the new symbol information will be fetched from the PLC. If you use the TPY file you must upload it to the webserver after compiling the PLC program and before you start the JavaScript clients. Otherwise data can be written to or fetched from wrong addresses. You can avoid this by using handles.

2. Instancing the Web Service Client

The heart of the library is the `WebServiceClient` constructor. You can create an instance by either using the „new“ keyword or the library function intended for this. Instancing the constructor you have to pass an object containing the arguments for the Web Service. Required are at least the URL of the `TcAdsWebService.dll` and the `AMS-NetID` of the PLC Runtime System. A minimal definition looks like this:

```
var PLC = TAME.WebServiceClient.createClient({
    serviceURL: "http://192.168.1.2/TcAdsWebService/TcAdsWebService.dll",
    amsNetId: "192.168.1.2.1.1",
    onReady: function() {
        //Put your stuff here
    }
});
```

Here is an example using more parameters:

```
var PLC = TAME.WebServiceClient.createClient({
    serviceURL: "http://192.168.1.2/TcAdsWebService/TcAdsWebService.dll",
    amsNetId: "192.168.1.2.1.1",
    amsPort: "802",
    alignment: „4“,
    language: en,
    onReady: function() {
        //Put your stuff here
    }
});
```

New in V4.x is the „onReady“ function. It has become necessary because the internal functions for getting the symbols are now based on asynchronous requests. All of your stuff that depends on the client should be started with this function.

You can also use the TPY file generated by TwinCAT. It contains all necessary information and you don't have to be worried about ports, alignment and so on. After compiling the PLC program the file must be copied to the webserver and the path specified in a parameter named „configFileUrl“:

```
var PLC = TAME.WebServiceClient.createClient({
    serviceURL: "http://192.168.1.2/TcAdsWebService/TcAdsWebService.dll",
    configFileUrl: " http://192.168.1.2/myproject.tpy",
    onReady: function() {
        //Put your stuff here
    }
});
```

The drawback is that after changing, compiling and uploading the PLC program you have to put the new TPY file on the webserver. You can reduce this by using a combination of the Upload and the TPY file. With the „forceUploadUsage“ parameter set to „true“ the symbol information of the Upload will be used and only the data type information will be extracted from the TPY file. So if you don't change data types or FB parameters the data type information should be the same. My tests were always fine but there is no guarantee that this always will be so.

After instantiating the WebServiceClient you have an object („PLC“ in the above examples) which provides the methods for sending read and write requests to the ADS WebService. They are described in the following chapters.

Parameters (Type, Required, Default)

amsNetId (string, required if „configFileUrl“ is not set, undefined) The ADS-AMSNetID of the PLC.
amsPort (string, optional, 801 802 803 804) The ADS-Port number of the Runtime-System.
alignment (string, optional, 1 4 8) The value is used for the automatic data alignment of structures. TwinCAT 2, x86: 1 Byte alignment TwinCAT 2, ARM: 4 Byte alignment TwinCAT 3: 8 Byte alignment
configFileUrl (string, optional, undefined) The path to the TPY file on the webserver. With this parameter set the symbol information, the data type information and informations regarding the PLC (NetId, alignment, port) will be extracted from the TPY file. The file is generated by TwinCAT everytime a PLC program is compiled and has to be copied to the webserver.
dontFetchSymbols (boolean, optional, undefined) When instantancing the client the symbols are fetched from the PLC. This is necessary to build the internal symbol table for the access to PLC variables by name. If this option is set to „true“ variables can not be accessed by name unless you load the symbols from a JSON string after the client is ready.
forceUploadUsage (boolean, optional, undefined) In combination with „configFileUrl“: If this parameter is set to „true“ the symbol table will not be built from the data of the *.tpy file, the Upload will be used instead. Only the data type information will be extracted from the file.
language (string, optional, ge en) Set the language for names of days and months. This option is used for the formatted output of dates.
onReady (function, required, undefined) This function will be called if the client is ready.
servicePassword (string, optional, null) The password of the service if the webserver requires authentication.

serviceUrl (string, required, undefined) The URL of the TcAdsWebservice.dll.
serviceUser (string, optional, null) The user of the service if the webserver requires authentication.
skipMissingTypes (boolean, optional, undefined) A few TwinCAT libraries lacks some definitions of the data types they include. While reading the types from the TPY file there will be an error and TAME stops importing. If you want to override this behavior you can set this parameter to „true“. If there is a missing type TAME will just drop an error to the console and continue importing. But be aware that you can't access PLC variables which use this data type except you provide the information manually.
syncXmlHttp (boolean, optional, undefined) Synchronous XMLHttpRequests are used by default if this option is set to „true“. This means that a script waits until a request is finished. The option can be overridden by the „sync“ parameter of a request. Intended for easier scripting in Tasker.
useHandles (boolean, optional, undefined) If this option is set to „true“ handles will be used for all requests. You have to get the handles from the PLC before you can access any symbol.

Instance Properties (Type, Default)

adsState (number, null)

The ADS state of the PLC as a number. According to my tests the state can only be 5 or 6, other ADS states return errors. You have to update the state with the „readAdsState()“ method (see chapter 9.2). If the readAdsState method returns an error the property will be set to *null*.

- 0: Invalid
- 1: Idle
- 2: Reset
- 3: Init
- 4: Start
- 5: Run
- 6: Stop
- 7: SaveConfig
- 8: LoadConfig
- 9: PowerFailure
- 10: PowerGood
- 11: Error
- 12: Shutdown
- 13: Suspend
- 14: Resume
- 15: Config
- 16: Reconfig
- 17: Maxstates

adsStateTxt (string, empty)

The ADS state as text. You have to update the state with the „readAdsState()“ method (see chapter 9.2). The string can have the values „Run“ or „Stop“. If the readAdsState method returns an error the string will be empty.

deviceState (number, null)

The PLC device state as a number. You have to update the state with the „readAdsState()“ method (see chapter 9.2). If the readAdsState method returns an error the property will be set to *null*.

- 0: Run
- 1: Stop

dateNames (object)

Contains an object with 4 arrays for the short and the full names of days and months used for the formatted output of dates. The language of the names can be set by the parameter „language“. If you need a language other than German or English, you can overwrite this property with your own names after instancing instead of changing the source code of the library.

maxDropReq (number, 10)
The maximum number of dropped requests in conjunction with the „id“-parameter of the requests. When the number has been reached a new request is fired and the counter starts again.
maxStringLen (number, 255)
The maximum length of strings.
useCheckBounds (boolean, true)
If set to „true“, the limits of numeric variables are checked before sending them to the PLC. If a limit is exceeded, the value will be set to the limit and an error message is written to the console. This applies to the following data types: BYTE, WORD , DWORD, USINT, SINT, UINT, INT, INT1DP, UDINT, DINT, TIME, REAL and LREAL.
dataTypeTableReady (boolean, true)
„True“ if the Data Type Table was successfully built with the information of the TPY file.
symbolTableReady (boolean, true)
„True“ if the Symbol Table was successfully built with the information of the Symbol Upload or the TPY file.
handleCacheReady (boolean, true)
„True“ if the Handle Cache was successfully built with a „getHandles()“ request..
xmlHttpRequestTimeout (number, 5000)
Timeout value for XMLHttpRequest.

3. ReadReq and WriteReq

There are 2 basic methods of the WebServiceClient: The Read Request (readReq) for fetching data from the PLC and the Write Request (writeReq) for sending data to the PLC. Both functions have one parameter only: an object I named Request Descriptor. They were the first methods implemented, all the other request functions (besides the sum requests) are just a kind of shortcuts and call them passing an internally created request descriptor. But there is an disadvantage: If you want to read or write multiple PLC variables they have to be set to fixed addresses. Therefore I recommend to **use the new sum commands for reading multiple variables (chapter 4)**. But if your device does not run the required TwinCAT versions you have to use the methods described in this chapter.

3.1 Read Request

Lets' use the instance of the WebServiceClient created in chapter 1. Assuming you want to read 3 PLC variables defined in a direct sequence, one of type integer at %MB53 and the other ones of type boolean at %MB55 and %MB56, a simple request definition looks like this:

```
PLC.readReq({
    addr: "%MB53",
    seq: true,
    items: [{
        type: "INT",
        jvar: "myVar1",
    }, {
        type: "BOOL",
        jvar: "myVar2"
    }, {
        type: "BOOL",
        jvar: "myVar3"
    }]
})
```

If the variables aren't following each other in a direct sequence, you have to set the address for each item separately. The request gets the whole data out of the PLC's memory, from the specified start address to the address of the last variable plus its length. So it's a bad idea to leave too much space between the addresses.

```
PLC.readReq({
  addr: "%MB53",
  items: [{
    addr: 53,
    type: "INT",
    jvar: "myVar1"
  }, {
    addr: 59,
    type: "BOOL",
    jvar: "myVar2"
  }, {
    addr: 60,
    type: "BOOL",
    jvar: "myVar3"
  }]
});
```

As you can see, the boolean variables are set to „%MB“ addresses. You can't access multiple boolean variables with „%MX“ addresses. I recommend to use the new SumReadRequest for this.

Supported Data Types

Data Type	Note
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	The JavaScript variables are multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	
DINT	
TIME	With a formatting information added the time value is converted to a formatted string, otherwise the output is a string with a value in milliseconds. See chapter 9.
TOD	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
DT	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
DATE	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
REAL	A value can be added to set the number of the decimal places. See chapter 9.
LREAL	A value can be added to set the number of the decimal places. See chapter 9.
STRING	

Parameters (Type, Required, Default)

addr (string, required, undefined)
The start address of the data to be read.
debug (boolean, optional, false)
If this option is set to true all internal used information of the request is written to the console.
id (number, optional, undefined)
The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of it's own. So if a requests definition is called in rapid succession one ore more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stucking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
items (array, required, undefined)
An array of objects containing the allocation of PLC and JavaScript variables (addresses, names and types).
oc (function, optional, undefined)
A function wich will be executed after the request has finished sucessfully.
ocd (number, optional, undefined)
A value of milliseconds for delaying the on-complete-function.
oe (function, optional, undefined)
A function wich will be executed after the request has finished with an error.
seq (boolean, optional, undefined)
If the variables in the PLC follow each other in a direct sequence, you don't have to specify the variable address of each one. You can omit the address parameters of the items and set this option to true instead.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

Item Parameters (Type, Required, Default)**addr** (number, required if the „seq“ parameter is false or undefined, undefined)

The address of the PLC variable.

type (string, required, undefined)

The type of the PLC variable. Can also contain a formatting part for some types. Look at chapter 9 for more information.

jvar (string, required, undefined)

The name of the JavaScript variable to store the data in. This must be a **global** variable. Namespaces of any depth and arrays are supported, i.e. "myNameSpace.myVars.var1" or "myNameSpace.myArray[0]".

prefix (string, optional, undefined)

An optional string which can be added to the data. The data value will be converted to a string with the prefix inserted **before** the value.

suffix (string, optional, undefined)

An optional string which can be added to the data. The data value will be converted to a string with the suffix inserted **after** the value.

3.2 Write Request

Sending values to the PLC is also quite easy. The parameters of the request descriptor are almost the same as of the read request. But the addresses of the PLC variables have always to be in a direct sequence. In the example below 4 variables are written, beginning at MB105.

```
PLC.writeReq({
  addr: "%MB105",
  items: [{
    type: "REAL",
    val: 234.12
  }, {
    type: "STRING.10",
    val: "Hello"
  }, {
    type: "BOOL",
    val: myVar3
  }, {
    type: "BOOL",
    val: false
  }]
});
```

Note that you can't access multiple variables with „%MX“ addresses using this method. It's only possible to write one single variable this way and I recommend the „writeBool“ method for this.

Supported Data Types

Data Type	Note
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	The JavaScript variables are multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	
DINT	
TIME	Without formatting the value must be specified in milliseconds. See chapter 9 for formatting parameters.
TOD	The value has to be a JS date object or a string (i.e. „21:33“).
DT	The value has to be a JS date object.
DATE	The value has to be a JS date object.
REAL	
LREAL	
STRING	

Parameters (Type, Required, Default)

addr (string, required, undefined)
The start address of the data to be written.
debug (boolean, optional, false)
If this option is set to true all internal used information of the request is written to the console.
id (number, optional, undefined)
The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of it's own. So if a requests definition is called in rapid succession one ore more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stucking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
items (array, required, undefined)
An array of objects containing the allocation of PLC and JavaScript variables (addresses, names and types).
oc (function, optional, undefined)
A function wich will be executed after the request has finished sucessfully.
ocd (number, optional, undefined)
A value of milliseconds for delaying the on-complete-function.
oe (function, optional, undefined)
A function wich will be executed after the request has finished with an error.
sync (boolean, optional, undefined)
A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

Item Parameters (Type, Required, Default)**type** (string, required, undefined)

The type of the PLC variable. Can also contain a formatting part for some types. Look at chapter 9 for more information.

val (boolean|number|string, required, undefined)

The value to send to the PLC. Can also be specified as a variable (even a local definend one).

4. Sum Commands

The sum commands are one of the new features of TAME 3. Together with the access to variables by name the usage is simple: Just build a list with the names of the required PLC variables. You don't need to specify types or string lengths, but you can override them manually, i.e. to use INT1DP instead of INT. The names of global PLC variables start with a point, the name of local PLC variables with the name of the instance they are defined in. They are case insensitive, „Main.IntVar“ is equal to „MAIN.INTVAR“ (this applies to all requests). Cause the sum commands are splitted in single ADS calls in the PLC you should not read more then 500 variables with one request. **Access to variables is only possible by names, not by addresses.**

4.2 Sum Read Request

Your PLC device must have TwinCAT V2.10 Build >= 1324 running to use this type of request. Time and date variables accept the „format“-option, REAL and LREAL the „dp“-option. Arrays, structures and arrays of structures are supported.

```
PLC.sumReadReq({
    items: [{
        name: ".GlobalBoolVar",
        jvar: "myVar1"
    }, {
        name: ".GlobalRealVar",
        dp: 2,
        jvar: "myVar2"
    }, {
        name: "MAIN.DateVar",
        format: "#DD#.#MM#.#YY#, #hh#:#mm",
        jvar: "myVar3"
    }, {
        name: "MAIN.IntVar",
        type: "INT1DP",
        jvar: "myVar4"
    }
});
```

An example using arrays and structures. For more information please refer to the chapters of reading arrays and structures.

```
var structdef = {
    var_1: "BYTE",
    var_2: "REAL.2",
    var_3: "SINT"
};

PLC.sumReadReq({
    items: [{
        name: ".GlobalDateArrayVar",
        format: "#DD",
        jvar: "myDateArrayVar"
    }, {
        name: ".GlobalStructVar",
        def: structdef,
        jvar: "myStructVar"
    }, {
        name: "MAIN.StructArrayVar",
        def: structdef,
        jvar: "myStructArrayVar"
    }
    ]
});
```

Supported Data Types

Data Type	Note
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	An INT converted to a variable with 1 decimal place, i.e. 637 in the PLC is 63.7 in the JavaScript variable.
UDINT	
DINT	
TIME	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a string with a value in milliseconds.
TOD	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
DT	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
DATE	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
REAL	Has a special formatting parameter „dp“ to set the number of the decimal places. For issues with converting REAL variables see chapter 10.1.
LREAL	Has a special formatting parameter „dp“ to set the number of the decimal places.
STRING	
User defined types	Arrays, structures and arrays of structures are supported.

Parameters (Type, Required, Default)

debug (boolean, optional, false)

If this option is set to true all internal used information of the request is written to the console.

id (number, optional, undefined)

This applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of its own. So if a request definition is called in rapid succession one or more requests can be fired before the previous one has finished. That doesn't make sense and can lead to sticking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.

items (array, required, undefined)

An array of objects containing the allocation of PLC and JavaScript variables (addresses, names and types).

oc (function, optional, undefined)

A function which will be executed after the request has finished successfully.

ocd (number, optional, undefined)

A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)

A function which will be executed after the request has finished with an error.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequest is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client definition.

Common Item Parameters (Type, Required, Default)

name (string, required, undefined) The name of the PLC variable.
type (string, optional, undefined) The type of the PLC variable. Can also contain a formatting part for some types. Look at chapter 9 for more information. The type is detected automatically, set it only if you know what you are doing.
jvar (string, required, undefined) The name of the JavaScript variable to store the data in. This must be a global variable. Namespaces of any depth and arrays are supported, i.e. "myNameSpace.myVars.var1" or "myNameSpace.myArray[0]".
handle (boolean, optional, undefined) Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
prefix (string, optional, undefined) An optional string which can be added to the data. The data value will be converted to a string with the prefix inserted before the value.
suffix (string, optional, undefined) An optional string which can be added to the data. The data value will be converted to a string with the suffix inserted after the value.

Type Dependent Item Parameters (Type, Required, Default)

dp | decPlaces (number, optional, undefined)

For data types REAL and LREAL: The number of decimal places. For issues with converting REAL variables see chapter 10.1.

def (object, required if the PLC variable is a structure or an array of structures, undefined)

For structures and arrays of structures: An object containing information about the structure definition. See chapter 7.

format (string, optional, undefined)

For data types TIME, TOD, DT, DATE: A string containing information for the formatted output of date and/or time. If used, the output is a string instead of a date object. See chapters 10.3 and 10.4 for this.

strlen (number, optional, undefined)

For data type STRING: The defined length of the string in the PLC. The value is detected automatically, set it only if you know what you are doing.

4.2 Sum Write Request

The Sum Write Request is supported by TAME 3.3. Your PLC device must have TwinCAT V2.11 Build >= 1550 running to use this type of request. Arrays, structures and arrays of structures are supported.

```
PLC.sumWriteReq({
  items: [{
    name: ".GlobalBoolVar",
    val: true
  }, {
    name: ".GlobalRealVar",
    val: 3.1223
  }, {
    name: "MAIN.TimeVar",
    format: "#h",
    val: 5
  }, {
    name: "MAIN.IntVar",
    type: "INT1DP",
    val: 123.3
  }, {
    name: ".GlobalStringVar",
    val: "This is a test"
  }
});
```

Supported Data Types

Data Type	Note
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	An INT converted to a variable with 1 decimal place, i.e. 637 in the PLC is 63.7 in the JavaScript variable.
UDINT	
DINT	
TIME	Without formatting the value must be specified in milliseconds. See chapter 9 for formatting parameters.
TOD	The value has to be a JS date object or a string (i.e. „21:33“).
DT	The value has to be a JS date object.
DATE	The value has to be a JS date object.
REAL	
LREAL	
STRING	
User defined types	Arrays, structures and arrays of structures are supported.

Parameters (Type, Required, Default)

debug (boolean, optional, false)

If this option is set to true all internal used information of the request is written to the console.

id (number, optional, undefined)

The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of its own. So if a requests definition is called in rapid succession one or more requests can be fired before the previous one has finished. That doesn't make sense and can lead to sticking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.

items (array, required, undefined)

An array of objects containing the allocation of PLC and JavaScript variables (addresses, names and types).

oc (function, optional, undefined)

A function which will be executed after the request has finished successfully.

ocd (number, optional, undefined)

A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)

A function which will be executed after the request has finished with an error.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequest is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client definition.

Common Item Parameters (Type, Required, Default)

name (string, required, undefined)
The name of the PLC variable.
type (string, optional, undefined)
The type of the PLC variable. Can also contain a formatting part for some types. Look at chapter 9 for more information. The type is detected automatically, set it only if you know what you are doing.
handle (boolean, optional, undefined)
Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
val (boolean number string, required, undefined)
The value to send to the PLC. Can also be specified as a variable (even a local definend one).

Type Dependent Item Parameters (Type, Required, Default)

def (object, required if the PLC variable is a structure or an array of structures, undefined)
For structures and arrays of structures: An object containing information about the structure definition. See chapter 7.
format (string, optional, undefined)
For data type TIME: A string containing information of the time base (i.e. „#m for minutes“).
strlen (number, optional, undefined)
For data type STRING: The defined lenght of the string in the PLC. The value is detected automatically, set it only if you know what you are doing.

5. Type Dependent Requests

For accessing single PLC variables, the library provides read and write requests for each supported data type.

5.1 Read Requests

Except for some special formatting options, the API of the requests is the same for all data types.

An example of reading a boolean variable defined in the MAIN program:

```
PLC.readBool({  
    name: "MAIN.BoolVar",  
    jvar: "myVar"  
});
```

I have tried to keep the syntax short, so it's no problem to write the commands in a single line:

```
PLC.readBool({name: "MAIN.BoolVar", jvar: "myVar"});
```

An example of reading a global string variable:

```
PLC.readString({  
    name: ".StringVar",  
    jvar: "myVar"  
});
```


An example of reading a PLC REAL value with rounding it to 1 decimal place. 100 ms after completing the request an alert window will be opened for displaying the value.

```
PLC.readReal({  
    name: "MAIN.RealVar",  
    dp: 1,  
    jvar: "myVar",  
    ocd: 100,  
    oc: function() {  
        alert("Value: " + myVar);  
    }  
});
```

Supported Data Types

PLC Data Type	Method	Note
BOOL	readBool	
BYTE	readByte	Basically the same as „readUsint“.
WORD	readWord	Basically the same as „readUint“.
DWORD	readDword	Basically the same as „readUdint“.
USINT	readUsint	
SINT	readSint	
UINT	readUint	
INT	readInt	
INT	readInt1Dp	An INT converted to a variable with 1 decimal place, i.e. 637 in the PLC is 63.7 in the JavaScript variable.
UDINT	readUdint	
DINT	readDint	
TIME	readTime	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a string with a value in milliseconds.
TOD	readTod	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
DT	readDt	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
DATE	readDate	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
REAL	readReal	Has a special formatting parameter „dp“ to set the number of the decimal places. For issues with converting REAL variables see chapter 10.1.
LREAL	readLreal	Has a special formatting parameter „dp“ to set the number of the decimal places.
STRING	readString	

Common Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter is not set, undefined)
The address of the PLC variable, can be used to read variables with fixed addresses. However, I recommend to use the „name“ parameter instead
name (string, required if the „addr“ parameter is not set, undefined)
The name of the PLC variable.
debug (boolean, optional, false)
If this option is set to true all internal used information of the request is written to the console.
id (number, optional, undefined)
This applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of its own. So if a request definition is called in rapid succession one or more requests can be fired before the previous one has finished. That doesn't make sense and can lead to sticking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
jvar (string, required, undefined)
The name of the JavaScript variable to store the data in. This must be a global variable. Namespaces of any depth and arrays are supported, i.e. "myNameSpace.myVars.var1" or "myNameSpace.myArray[0]".
handle (boolean, optional, undefined)
Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
oc (function, optional, undefined)
A function which will be executed after the request has finished successfully.
ocd (number, optional, undefined)
A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)
A function which will be executed after the request has finished with an error.
prefix (string, optional, undefined)
An optional string which can be added to the data. The data value will be converted to a string with the prefix inserted before the value.
suffix (string, optional, undefined)
An optional string which can be added to the data. The data value will be converted to a string with the suffix inserted after the value.
sync (boolean, optional, undefined)
A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

Type Dependent Parameters (Type, Required, Default)

dp decPlaces (number, optional, undefined)
For data types REAL and LREAL: The number of decimal places. For issues with converting REAL variables see chapter 10.1.
format (string, optional, undefined)
For data types TIME, TOD, DT, DATE: A string containing information for the formatted output of date and/or time. If used, the output is a string instead of a date object. See chapters 10.3 and 10.4 for this.
strlen (number, required if the „addr“ parameter is used, undefined)
For data type STRING: The defined length of the string in the PLC. If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“-parameter.

5.2 Write Requests

An example of writing an integer:

```
PLC.writeInt({  
    name: "MAIN.IntVar",  
    val: 2635  
});
```

Set a global boolean PLC variable to „true“ and 1 second later to „false“:

```
PLC.writeBool({  
    name: ".BoolVar",  
    val: true,  
    ocd: 1000,  
    oc: function() {  
        PLC.writeBool({name: ".boolVar", val: false});  
    }  
});
```

Write a time variable with a value of 15 minutes:

```
PLC.writeTIME({  
    name: ".TimeVar",  
    format: "#m",  
    val: 15  
});
```

Supported Data Types

PLC Data Type	Method	Note
BOOL	writeBool	
BYTE	writeByte	Basically the same as „writeUsint“.
WORD	writeWord	Basically the same as „writeUint“.
DWORD	writeDword	Basically the same as „writeUdint“.
USINT	writeUsint	
SINT	writeSint	
UINT	writeUint	
INT	writeInt	
INT	writeInt1Dp	The JavaScript variable is multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	writeUdint	
DINT	writeDint	
TIME	writeTime	Has a special parameter „format“. See chapter 10.3 for this.
TOD	writeTod	The value has to be a JS date object or a string (i.e. „21:33“).
DT	writeDt	The value has to be a JS date object.
DATE	writeDate	The value has to be a JS date object.
REAL	writeReal	For issues with converting REAL variables see chapter 10.1.
LREAL	writeLreal	
STRING	writeString	

Common Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter is not set, undefined) The address of the PLC variable, can be used to write variables with fixed addresses. However, I recommend to use the „name“ parameter instead
name (string, required if the „addr“ parameter is not set, undefined) The name of the PLC variable.
debug (boolean, optional, false) If this option is set to true all internal used information of the request is written to the console.
id (number, optional, undefined) This applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of its own. So if a request definition is called in rapid succession one or more requests can be fired before the previous one has finished. That doesn't make sense and can lead to sticking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
handle (boolean, optional, undefined) Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
oc (function, optional, undefined) A function which will be executed after the request has finished successfully.
ocd (number, optional, undefined) A value of milliseconds for delaying the on-complete-function.
oe (function, optional, undefined) A function which will be executed after the request has finished with an error.

val (boolean|number|string, required, undefined)

The value to send to the PLC. Can also be specified as a variable (even a local definend one).

sync (boolean, optional, undefined)

A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

Type Dependent Parameters (Type, Required, Default)

strlen (number, required if the „addr“ parameter is used, undefined)

For data type STRING: The defined lenght of the string in the PLC. If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“-parameter.

6. Arrays

There are some powerful methods for accessing arrays and structures in the PLC. This way you can read or write thousands of values with one single command. Just like for the single variables, there are reading and writing methods for each data type.

6.1 Read Requests

An example of reading an array of strings. Other than TAME V2 this version detects the defined length of arrays and strings automatically. The parameters „arlen“ and „strlen“ can be omitted and the command just looks like this:

```
PLC.readArrayOfString({  
    name: ".StringArray",  
    jvar: "myArray"  
});
```

An example of reading an array of 10 DATE values. After executing, the JavaScript array will contain formatted strings.

```
PLC.readArrayOfDate({  
    name: "MAIN.DateArray",  
    format: "#WEEKDAY#, #DD#.#MM#.#YYYY",  
    jvar: "myArray"  
});
```

Supported Data Types

PLC Data Type	Method	Note
BOOL	readArrayOfBool	
BYTE	readArrayOfByte	Basically the same as „readArrayOfUsint“.
WORD	readArrayOfWord	Basically the same as „readArrayOfUint“.
DWORD	readArrayOfDword	Basically the same as „readArrayOfUdint“.
USINT	readArrayOfUsint	
SINT	readArrayOfSint	
UINT	readArrayOfUint	
INT	readArrayOfInt	
INT	readArrayOfInt1Dp	An INT converted to a variable with 1 decimal place, i.e. 637 in the PLC is 63.7 in the JavaScript variable.
UDINT	readArrayOfUdint	
DINT	readArrayOfDint	
TIME	readArrayOfTime	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a string with a value in milliseconds.
TOD	readArrayOfTod	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
DT	readArrayOfDt	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
DATE	readArrayOfDate	With the „format“-parameter set the time value is converted to a formatted string, otherwise the output is a date object.
REAL	readArrayOfReal	Has a special formatting parameter „dp“ to set the number of the decimal places. For issues with converting REAL variables see chapter 10.1.
LREAL	readArrayOfLreal	Has a special formatting parameter „dp“ to set the number of the decimal places.

STRING	readArrayOfString	
--------	-------------------	--

Common Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter is not set, undefined) The address of the PLC variable, can be used to read variables with fixed addresses. However, I recommend to use the „name“ parameter instead.
name (string, required if the „addr“ parameter is not set, undefined) The name of the PLC variable.
arrlen (number, required if the „addr“ parameter is used, undefined) The length of the array (the number of the items). If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“-parameter.
debug (boolean, optional, false) If this option is set to true all internal used information of the request is written to the console.
id (number, optional, undefined) This applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of its own. So if a request definition is called in rapid succession one or more requests can be fired before the previous one has finished. That doesn't make sense and can lead to sticking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
jvar (string, required, undefined) The name of the JavaScript variable to store the data in. This must be a global variable. Namespaces of any depth and arrays are supported, i.e. "myNameSpace.myVars.var1" or "myNameSpace.myArray[0]".
handle (boolean, optional, undefined) Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
oc (function, optional, undefined) A function which will be executed after the request has finished successfully.

ocd (number, optional, undefined)

A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)

A function which will be executed after the request has finished with an error.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

Type Dependent Parameters (Type, Required, Default)

dp | decPlaces (number, optional, undefined)

For data types REAL and LREAL: The number of decimal places. For issues with converting REAL variables see chapter 10.1.

format (string, optional, undefined)

For data types TIME, TOD, DT, DATE: A string containing information for the formatted output of date and/or time. If used, the output is a string instead of a date object. See chapters 10.3 and 10.4 for this.

strlen (number, required if the „addr“ parameter is used, undefined)

For data type STRING: The defined length of the string in the PLC. If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“ parameter.

6.2 Write Requests

An example of writing an array of boolean variables:

```
PLC.writeArrayOfBool({  
    name: "MAIN.BoolArray",  
    val: myArray  
});
```

You can also write only one item of the array. Here an example of writing the last element of an array of 10 integer variables:

```
PLC.writeArrayOfInt({  
    name: "MAIN.IntArray",  
    item: 9,  
    val: myArray  
});
```

Supported Data Types

PLC Data Type	Method	Note
BOOL	writeArrayOfBool	
BYTE	writeArrayOfByte	Basically the same as „writeArrayOfUsint“.
WORD	writeArrayOfWord	Basically the same as „writeArrayOfUint“.
DWORD	writeArrayOfDword	Basically the same as „writeArrayOfUdint“.
USINT	writeArrayOfUsint	
SINT	writeArrayOfSint	
UINT	writeArrayOfUint	
INT	writeArrayOfInt	
INT	writeArrayOfInt1Dp	The JavaScript variables are multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	writeArrayOfUdint	
DINT	writeArrayOfDint	
TIME	writeArrayOfTime	Has a special parameter „format“. See chapter 10.3 for this.
TOD	writeArrayOfTod	The values have to be JS date objects or strings (i.e. „21:33“).
DT	writeArrayOfDt	The values have to be JS date objects.
DATE	writeArrayOfDate	The values have to be JS date objects.
REAL	writeArrayOfReal	For issues with converting REAL variables see chapter 10.1.
LREAL	writeArrayOfLreal	
STRING	writeArrayOfString	

Common Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter is not set, undefined) The address of the PLC variable, can be used to read variables with fixed addresses. However, I recommend to use the „name“ parameter instead
name (string, required if the „addr“ parameter is not set, undefined) The name of the PLC variable.
arrlen (number, required if the „addr“ parameter is used, undefined) The length of the array (the number of the items). If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“-parameter.
debug (boolean, optional, false) If this option is set to true all internal used information of the request is written to the console.
id (number, optional, undefined) This applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of its own. So if a request definition is called in rapid succession one or more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stalling. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
item (number, optional, undefined) If this parameter is set to the index number (starting with 0) of an array item, only this item will be written to the PLC.
handle (boolean, optional, undefined) Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
oc (function, optional, undefined) A function which will be executed after the request has finished successfully.

oed (number, optional, undefined)
A value of milliseconds for delaying the on-complete-function.
oe (function, optional, undefined)
A function which will be executed after the request has finished with an error.
val (array, required, undefined)
The array of values to send to the PLC.
sync (boolean, optional, undefined)
A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

Type Dependent Parameters (Type, Required, Default)

strlen (number, required if the „addr“ parameter is used, undefined)
For data type STRING: The defined length of the string in the PLC. If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“ parameter.

7. Structures

Accessing structures is somewhat more complex. The request needs information about the elements of the structure in the PLC's memory. If you use the TPY file the information will be automatically read from the file. If you don't use the file or want to use your own property names you have to create your own definition. For this the descriptor has the „def“ parameter. It requires an object literal containing the names of the properties of the storing JavaScript object and the corresponding data types.

Assuming you have a user defined data type like this:

```
TYPE ST_Test :  
    STRUCT  
        nByte: BYTE;  
        fReal: REAL;  
        iSint: SINT;  
        tTime: TIME;  
        arrString: ARRAY[1..5] OF STRING(6);  
        iInt: INT;  
    END_STRUCT  
END_TYPE
```

And a global variable of this type named „TestStruct:

```
TestStruct: ST_Test;
```

The structure definition could look like this. You can set the names of the properties as you need. The order of the data types has to be the same as in the PLC.

```
var structdef = {  
    var_1: "BYTE",  
    var_2: "REAL.2",  
    var_3: "SINT",  
    var_4: "TIME",  
    var_5: "ARRAY.5.STRING.6", //Array of 5 strings with 6 chars  
    var_6: "INT1DP"  
};
```

Note that you can use one definition for read and write requests. If used with a write request, all formatting options will be ignored, except of the parameter for the length of strings. See Chapter 9 for examples of type dependent formatting in structure definitions.

7.1 Read Request

Of course you have to define a JavaScript object to store the data, the properties of the object contain the values. For read requests normally you haven't to define the properties, they will be created if the request is executed. So instead of:

```
var myVar = {  
    var_1: 0,  
    var_2: 0,  
    var_3: 0,  
    var_4: new Date(),  
    var_5: [],  
    var_6: 0  
};
```

You could define the object just like this:

```
var myVar = {};
```

The request again is quite simple:

```
PLC.readStruct({  
    name: ".TestStruct",  
    def: structdef,  
    jvar: "myVar"  
});
```

Sometimes it can be useful to store the data in single properties of the object instead of an array. For this there is a special option „SP“. It must be the last option of an array definition. If we use the example above the definition looks like this:

```
var structdef = {  
    var_1: "BYTE",  
    var_2: "REAL.2",  
    var_3: "SINT",  
    var_4: "TIME",  
    var_5: "ARRAY.5.STRING.6.SP", //Array of 5 strings with 6 chars  
    var_6: "INT1DP"  
};
```

The JavaScript object now contains 5 single properties with strings instead of one property with an array of strings. The array index is just added to the name of the properties.

```
var myVar = {  
    var_1: 0,  
    var_2: 0,  
    var_3: 0,  
    var_4: new Date(),  
    var_50: "",  
    var_51: "",  
    var_52: "",  
    var_53: "",  
    var_54: "",  
    var_6: 0  
};
```

Supported Data Types

Data Type	Note
ARRAY	
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	The JavaScript variables are multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	
DINT	
TIME	With a formatting information added the time value is converted to a formatted string, otherwise the output is a string with a value in milliseconds. See chapter 9.
TOD	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
DT	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
DATE	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
REAL	A value can be added to set the number of the decimal places. See chapter 9.
LREAL	A value can be added to set the number of the decimal places. See chapter 9.
STRING	

Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter in not set, undefined)
The address of the PLC variable, can be used to read variables with fixed addresses. However, I recommend to use the „name“ parameter instead
name (string, required if the „addr“ parameter in not set, undefined)
The name of the PLC variable.
debug (boolean, optional, false)
If this option is set to true all internal used information of the request is written to the console.
def (object, required, undefined)
An object containing information about the structure definition.
id (number, optional, undefined)
The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of it's own. So if a requests definition is called in rapid succession one ore more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stucking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
jvar (string, required, undefined)
The name of the JavaScript variable to store the data in. This must be a global variable. Namespaces of any depth and arrays are supported, i.e. "myNameSpace.myVars.var1" or "myNameSpace.myArray[0]".
handle (boolean, optional, undefined)
Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
oc (function, optional, undefined)
A function wich will be executed after the request has finished sucessfully.

oed (number, optional, undefined)

A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)

A function which will be executed after the request has finished with an error.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

7.2 Write Request

For write requests applies the same as for read requests. The difference is, of course, that the JavaScript object and its properties must exist before the request is executed.

```
var myVar = {  
    var_1: 12,  
    var_2: 7464.313,  
    var_3: -96,  
    var_4: new Date("January 3, 2012 18:54:00"),  
    var_5: ["This", "is", "a", "string", "array"],  
    var_6: 34.5  
};
```

Like the read request, the write request is also quite simple:

```
PLC.writeStruct({  
    name: ".TestStruct",  
    def: structdef,  
    val: myVar  
});
```

Supported Data Types

Data Type	Note
ARRAY	
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	The JavaScript variables are multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	
DINT	
TIME	Without formatting the value must be specified in milliseconds. See chapter 9 for formatting parameters.
TOD	The value has to be a JS date object or a string (i.e. „21:33“).
DT	The value has to be a JS date object.
DATE	The value has to be a JS date object.
REAL	
LREAL	
STRING	

Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter in not set, undefined)
The address of the PLC variable, can be used to read variables with fixed addresses. However, I recommend to use the „name“ parameter instead
name (string, required if the „addr“ parameter in not set, undefined)
The name of the PLC variable.
debug (boolean, optional, false)
If this option is set to true all internal used information of the request is written to the console.
def (object, required, undefined)
An object containing information about the structure definition.
id (number, optional, undefined)
The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of it's own. So if a requests definition is called in rapid succession one ore more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stucking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
handle (boolean, optional, undefined)
Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.
oc (function, optional, undefined)
A function wich will be executed after the request has finished sucessfully.
ocd (number, optional, undefined)
A value of milliseconds for delaying the on-complete-function.
oe (function, optional, undefined)
A function wich will be executed after the request has finished with an error.

val (object, required, undefined)

The array of values to send to the PLC.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

8. Arrays Of Structures

The most powerful method's of the library can read or write a whole array of structures. They are a combination of the methods for accessing arrays and structures and you should read the previous chapters to understand what's going on. Based on the structure definition of chapter 7 here is the definition of a global PLC variable :

```
TestArray: ARRAY [1..2] OF ST_Test;
```

8.1 Read Request

The definition of the JavaScript array:

```
var myArray = [];
```

The request definition itself is quite simple again.

```
PLC.readArrayOfStruct({  
    name: ".TestArray",  
    def: structdef,  
    jvar: "myArray"  
});
```

Supported Data Types

Data Type	Note
ARRAY	
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	The JavaScript variables are multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	
DINT	
TIME	With a formatting information added the time value is converted to a formatted string, otherwise the output is a string with a value in milliseconds. See chapter 9.
TOD	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
DT	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
DATE	With a formatting information added time value is converted to a formatted string, otherwise the output is a date object.
REAL	A value can be added to set the number of the decimal places. See chapter 9.
LREAL	A value can be added to set the number of the decimal places. See chapter 9.
STRING	

Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter in not set, undefined)
The address of the PLC variable, can be used to read variables with fixed addresses. However, I recommend to use the „name“ parameter instead
name (string, required if the „addr“ parameter in not set, undefined)
The name of the PLC variable.
arrlen (number, required if the „addr“ parameter is used, undefined)
The length of the array (the number of the items). If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“-parameter.
debug (boolean, optional, false)
If this option is set to true all internal used information of the request is written to the console.
def (object, required, undefined)
An object containing information about the structure definition.
id (number, optional, undefined)
The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of it's own. So if a requests definition is called in rapid succession one ore more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stucking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
jvar (string, required, undefined)
The name of the JavaScript variable to store the data in. This must be a global variable. Namespaces of any depth and arrays are supported, i.e. "myNameSpace.myVars.var1" or "myNameSpace.myArray[0]".
handle (boolean, optional, undefined)
Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.

oc (function, optional, undefined) A function which will be executed after the request has finished successfully.
ocd (number, optional, undefined) A value of milliseconds for delaying the on-complete-function.
oe (function, optional, undefined) A function which will be executed after the request has finished with an error.
sync (boolean, optional, undefined) A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

8.2 Write Request

```
var myArray = [{  
    var_1: 12,  
    var_2: 7464.313,  
    var_3: -96,  
    var_4: new Date("January 3, 2012 18:54:00"),  
    var_5: ["This", "is", "a", "string", " array."],  
    var_6: 34.5  
},{  
    var_1: 109,  
    var_2: 873.40,  
    var_3: -5,  
    var_4: new Date(1325619660),  
    var_5: ["This", "is a", "string", " array", "too." ],  
    var_6: 122.9  
}];
```

```
PLC.writeArrayOfStruct({  
    name: ".TestArray",  
    def: structdef,  
    val: myArray  
});
```

Note that you also can use the „item“-parameter, if you want to write a single array item (one structure) only.

Supported Data Types

Data Type	Note
ARRAY	
BOOL	
BYTE	
WORD	
DWORD	
USINT	
SINT	
UINT	
INT	
INT1DP	The JavaScript variables are multiplied by 10 and converted to an integer, i.e. a value of 12.313 in JS is 123 in the PLC.
UDINT	
DINT	
TIME	Without formatting the value must be specified in milliseconds. See chapter 9 for formatting parameters.
TOD	The value has to be a JS date object or a string (i.e. „21:33“).
DT	The value has to be a JS date object.
DATE	The value has to be a JS date object.
REAL	
LREAL	
STRING	

Parameters (Type, Required, Default)

addr (string, required if the „name“ parameter in not set, undefined) The address of the PLC variable, can be used to read variables with fixed addresses. However, I recommend to use the „name“ parameter instead
name (string, required if the „addr“ parameter in not set, undefined) The name of the PLC variable.
arrlen (number, required if the „addr“ parameter is used, undefined) The length of the array (the number of the items). If the variable is accessed by name the value is detected automatically, it should only be set if you use the „addr“-parameter.
debug (boolean, optional, false) If this option is set to true all internal used information of the request is written to the console.
def (object, required, undefined) An object containing information about the structure definition.
id (number, optional, undefined) This applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of its own. So if a requests definition is called in rapid succession one or more requests can be fired before the previous one has finished. That doesn't make sense and can lead to sticking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.
item (number, optional, undefined) If this parameter is set to the index number (starting with 0) of an array item, only this item will be written to the PLC.
handle (boolean, optional, undefined) Instead of a direct access a handle will be used. For proper function the handle has to be fetched before. The option overrides by the „useHandles“ parameter of the client definition.

oc (function, optional, undefined) A function which will be executed after the request has finished successfully.
ocd (number, optional, undefined) A value of milliseconds for delaying the on-complete-function.
oe (function, optional, undefined) A function which will be executed after the request has finished with an error.
val (array, required, undefined) The array of values to send to the PLC.
sync (boolean, optional, undefined) A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client.

9. Handles

A handle is a code for a specific PLC symbol. Instead of a direct memory access via IndexGroup and IndexOffset one first have to get a handle for the symbol and use this for addressing a read/write request. This has 2 advantages: If the address of a symbol has changed after compiling the PLC program the handle is invalid and no data will be written to or read from the wrong address. The second is that it is easier to access elements of complex data types and internal variables and parameters of FB instances. There is also a drawback: You have to get the handles from the PLC before you can use them and they should be released if not needed anymore, i.e. while closing the webpage.

TAME provides several ways to access PLC variables by name. The Symbol Upload is used by default, the symbol information will automatically be fetched while the client starts. Alternatively the TPY file generated by TwinCAT can be used. While the Upload is automatically updated by TwinCAT during the upload of a PLC program the TPY has to be copied to webserver manually. All read or write requests without using a handle are based on a direct memory access via Index Group and Index Offset. As mentioned in the introduction it is possible to write data to or read from wrong addresses if a PLC program was uploaded without closing the client before.

Standard Data Types (BOOL, INT, WORD ...)		
	global variables / local variables of PRG's	local variables and parameters of FB instances / elements of structures or arrays
Upload (default)	no restrictions	not possible
Upload + Handles	no restrictions	type has to be defined by user
TPY file (with or without handles)	no restrictions	no restrictions

User Defined Data Types (Arrays/Structures)		
	global variables / local variables of PRG's	local variables and parameters of FB instances / elements of structures or arrays
Upload (default)	no restrictions	not possible
Upload + Handles	no restrictions	not possible (yet)
TPY file (with or without handles)	no restrictions	no restrictions

9.1 Getting Handles

To fetch handles from the PLC there is a method called „getHandles()“. You just have to provide an array of the symbol names for which the handles should be fetched. To ensure the handles are ready before using them you should start all functions with the „oc“ (on complete) function of the request. Check the „handles“ example for a better understanding.

```
Plc.getHandles({
  symbols: [
    '.In_Bool1', '.In_Bool2', '.IN_SINT1', '.IN_INT1', '.IN_DINT', '.IN_STRING',
    '.IN_TIME', '.IN_REAL', '.TOD_TEST', '.DT_TEST', '.DATE_TEST',
    'MAIN.fbRamp1.nRamp', 'MAIN.fbRamp2.nRamp'
  ],
  oc: function() {
    //Start requests depending on handles
  }
});
```

Because a sum command is used to get the handles a **TwinCAT version 2.11 >= Build 1550** is necessary. After successfully fetching the handles the „handleCacheReady“ property of the client will be set to „true“.

Parameters (Type, Required, Default)

debug (boolean, optional, false)

If this option is set to true all internal used information of the request is written to the console.

id (number, optional, undefined)

The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of it's own. So if a requests definition is called in rapid succession one ore more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stucking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.

symbols (array, required, undefined)

An array of symbol names for which the handles will be fetched.

oc (function, optional, undefined)

A function wich will be executed after the request has finished sucessfully.

ocd (number, optional, undefined)

A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)

A function wich will be executed after the request has finished with an error.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client definition.

9.2 Using Handles

There are 2 ways to specify if a request should use a handle. By setting the client parameter „useHandles“ to „true“ all requests are forced to use handles by default.

```
var PLC = TAME.WebServiceClient.createClient({
    serviceURL: "http://192.168.1.2/TcAdsWebService/TcAdsWebService.dll",
    amsNetId: "192.168.1.2.1.1",
    useHandles: TRUE,
    onReady: function() {
        //Put your stuff here
    }
});
```

If you want a specific request to use a handle just set the „handle“ parameter of this request to true.

```
PLC.readBool({name: "MAIN.BoolVar", handle: TRUE, jvar: "myVar"});
```

This also works for items of a sum commando.

```
PLC.sumReadReq({
    items: [{
        name: ".GlobalBoolVar",
        handle: TRUE,
        jvar: "myVar1"
    }, {
        name: ".GlobalRealVar",
        jvar: "myVar2"
    }
});
```


9.3 Releasing Handles

Handles which will not be used anymore should be released. Otherwise they stay in the memory of the PLC until the next restart. The „releaseHandles“ method accepts an array of symbol names for which the handles should be released.

```
Plc.releaseHandles({  
  symbols: [  
    'In_Bool1', 'In_Bool2', 'IN_SINT1', 'IN_INT1', 'IN_DINT', 'IN_STRING',  
    'IN_TIME', 'IN_REAL', 'TOD_TEST', 'DT_TEST', 'DATE_TEST',  
    'MAIN.fbRamp1.nRamp', 'MAIN.fbRamp2.nRamp'  
  ]  
});
```

Without the list all handles will be released.

```
Plc.releaseHandles();
```

After successfully releasing all handles the „handleCacheReady“ property of the client will be set to „false“.

Parameters (Type, Required, Default)

debug (boolean, optional, false)

If this option is set to true all internal used information of the request is written to the console.

id (number, optional, undefined)

The is applies only for asynchronous XMLHttpRequests (default): All requests are completely independent of one another. If you have a request defined and call it through a loop, every execution generates an AJAX request of it's own. So if a requests definition is called in rapid succession one ore more requests can be fired before the previous one has finished. That doesn't make sense and can lead to stucking. This option is an ID to have a marker for the otherwise „anonymous“ requests invoked with the same definition. Be aware that you don't give different request definitions the same number. If this parameter is set to a value (> 0) and the request is fired, all following requests are dropped until the current one has finished or the number set in the „maxDropReq“ property of the webservice instance has been reached. For every dropped request a message is written to the console. This option makes only sense with fast cyclic reading or writing and has less meaning with modern browsers. Just try it for yourself.

symbols (array, optional, undefined)

An array of symbol names for which the handles will be fetched.

oc (function, optional, undefined)

A function wich will be executed after the request has finished sucessfully.

ocd (number, optional, undefined)

A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)

A function wich will be executed after the request has finished with an error.

sync (boolean, optional, undefined)

A synchronous XMLHttpRequests is used if this option is set to „true“. This means that a script waits until the request is finished. The option overrides by the „syncXmlHttp“ parameter of the client definition.

9.4 Logging Handles

For test purposes you can print the cached handles in the console:

```
Plc.logHandleCache();
```

10. Other Methods

10.1 Import/Export the Symbol Table

With V3.2 there are 2 new methods for reading and writing the symbol table from/to a JS string variable. This was especially intended for Tasker scripts to help you write/read the symbols in/from a file. So you can store the data locally and it's not necessary to fetch them everytime you connect to the PLC.

Methods for Handling the Symbol Table (Type, Arguments, Return)

logSymbols (function, none, undefined)

With this method you can output the internal symbol table to the JavaScript console of your browser.

getSymbolsAsJSON (function, none, string)

This is a method to store the internal symbol table in a JSON string.

Example: `var jstr = PLC.getSymbolsAsJSON();`

setSymbolsFromJSON (function, string, undefined)

This is a method to build the internal symbol table from a JSON string.

Example: `PLC.setSymbolsFromJSON(jstr);`

10.2 Import/Export the Data Type Table

There are 2 methods for reading and writing the data type information from/to a JS string variable. The Data Type Table is built with the information of the TPY file. This was especially intended for Tasker scripts to help you write/read the Data Types in/from a file. So you can store the data locally and it's not necessary to fetch everytime you connect to the PLC.

Methods for Handling the Data Type Table (Type, Arguments, Return)

logDataTypes (function, none, undefined)

With this method you can output the internal symbol table to the JavaScript console of your browser.

getDataTypesAsJSON (function, none, string)

This is a method to store the internal symbol table in a JSON string.

Example: `var jstr = PLC.getDataTypesAsJSON();`

setDataTypesFromJSON (function, string, undefined)

This is a method to build the internal symbol table from a JSON string.

Example: PLC.setDataTypesFromJSON(jstr);

10.3 Check the PLC State

Methods for Reading the ADS State (Type, Arguments, Return)

readAdsState (function, none, undefined)

With this method you can check the current ADS-state of the PLC. The state is stored in 3 instance properties of the client (see chapter 2).

Parameters (Type, Required, Default)

oc (function, optional, undefined)

A function which will be executed after the request has finished successfully.

ocd (number, optional, undefined)

A value of milliseconds for delaying the on-complete-function.

oe (function, optional, undefined)

A function which will be executed after the request has finished with an error.

11.Type Dependent Parameters

11.1 REAL and LREAL

The data type REAL is a 32 bit numeric variable with one bit as algebraic sign, 8 bit for the exponent and 23 bit for the mantissa. Though a decimal floating point number can be converted to a binary floating point number, not every value can be represented due to the limited number of digits in the mantissa. In example, the decimal floating point number 0,1 will be converted to a value of 0.10000000149011612. Moreover, the data type FLOAT in JavaScript has a length of 64 bit but the type REAL in the PLC has a length of 32 bit.

For reading PLC REAL values there is a parameter for rounding the value to the desired number of places after the decimal point. I recommend to use it always. For writing REAL variables I have tried to find a fast and also accurate conversion, but I cannot rule out the possibility to get weird values while using those methods.

```
PLC.readReq({
  addr: "%MB53",
  seq: true,
  items: [{
    type: "REAL.1",
    jvar: "myVar"
  }]
});
```

For the method „readReq“ and those for reading structures just add the number of decimal places to the type, separated by a point:

Example of a structure definition for „readStruct“ and „readArrayOfStruct“:

```
var structdef = {
  myreal: "REAL.2"
};
```

For the methods „readReal“, „readArrayOfReal“ and the SumReadRequest there is the separate parameter „dp“ or „decPlaces“:

```
PLC.readReal({  
    name: "MAIN.RealVar",  
    dp: 2,  
    jvar: "myVar"  
});
```

The data type LREAL is a 64 bit numeric variable with one bit as algebraic sign, 11 bit for the exponent and 52 bit for the mantissa. Like the type REAL it can not represent every value of a decimal floating point number but it is much more precise. And there are no rounding errors when converting the values, cause both sides use the same format and length for storing the values in the memory. The parameter for rounding the value to the desired number of places after the decimal point is also supported.

11.2 STRING

A string in TwinCAT has a length of 80 characters by default. But you can change this and specify the length from 1 to 255 characters if you need.

This is a definition of a string in the PLC with 10 characters length:

```
MyString: STRING(10);
```

If you do so, you have to set the length in your JavaScript code too for some requests: for the ReadRequest (readReq) and the WriteRequest (writeReq) as well as for requests using structures. For all others the length can be detected automatically.

Just add the number to the type separated by a point:

```
PLC.readReq({
    addr: "%MB16",
    seq: true,
    items: [{
        type: "STRING.10",
        jvar: "myVar"
    }]
});
```

Example of a structure definition for „readStruct“ and „readArrayOfStruct“:

```
var structdef = {
    mystring: "STRING.10"
};
```

There is another thing you have to be aware of when you work with strings. Strings in TwinCAT are null-terminated, so the real length in memory is the specified length + 1 byte (11 bytes for the examples above, 81 bytes for a standard string). The library adds the termination automatically, so you don't need to worry about it. But you must pay attention when you use strings with fixed addresses and set the address of the next variable.

11.3 TIME

When you read a TIME value from the PLC you will get a string with a value in milliseconds, if no formatting parameter is used. For the methods „readReq“, „writeReq“ and those for reading/writing structures just add the formatting string to the type separated by a point, for all other requests use the „format“-parameter.

The separator for placeholders and custom characters/substrings is the „#“-character. At this separator the parser splits the string into pieces and searches for formatting instructions.

Example of a read request, it returns a value converted to hours. A value of 90 minutes in the PLC would be a value of "1.5" in the JavaScript variable.

```
PLC.readReq({
  addr: "%MB50",
  seq: true,
  items: [{
    type: "TIME.#h",
    jvar: "myVar"
  }]
});
```

For read requests you can use a combination of formatting instructions. A value of 92 minutes and 22 seconds would be a string of "1 - 32 - 22" in the JavaScript variable with the example below..

```
var structdef = {
  mytime: "TIME.#hh# - #mm# - #ss"
};
```

I've tried to keep the formatting flexible, so you can build whole sentences.

```
var structdef = {  
    mytime: "TIME.#Actual time: #hh# hours, #mm# minutes and #ss# seconds."  
};
```

For writing values to a PLC TIME variable without formatting you also have to use a value in milliseconds. But other than the write request you can use only one formatting instruction (i.e. minutes or hours).

An example of writing a value of 1200 ms:

```
PLC.writeTime({  
    name: ".TimeVar",  
    val: 1200  
});
```

Writing a value of 4 minutes:

```
PLC.writeTime({  
    name: ".TimeVar",  
    format: #m,  
    val: 4  
});
```

Formatting Instructions

Instruction	Return Value
none	milliseconds
dd	days, if the value is < 10 a leading zero will be added (i.e. 06)
d	days
hh	hours, if the value is < 10 a leading zero will be added.
h	hours
mm	minutes, if the value is < 10 a leading zero will be added.
m	minutes
ss	seconds, if the value is < 10 a leading zero will be added.
s	seconds
msmsms	milliseconds, 3 digits
ms	milliseconds

11.4 TOD, DT and DATE

Although the parameters and instructions are almost the same as of type TIME, the conversion is a bit different. For example, when you read a TOD value of 0 hours and 30 minutes from the PLC using a „#h“ formatting instruction, you will get a value of 0, with the type TIME it would be 0.5. If no formatting string is used, the methods for these three types return JavaScript date objects.

```
PLC.readReq({
  addr: "%MB250",
  seq: true,
  items: [{
    type: "TOD.#hh#:#mm",
    jvar: "myVar"
  }]
});
```

A read request with 2 variables:

```
PLC.readReq({
  addr: "%MB260",
  seq: true,
  items: [{
    type: "DT.#WEEKDAY#, #hh#:#mm",
    jvar: "myVar"
  }, {
    type: "DATE.#DD#, #MONTH# #YYYY",
    jvar: "myVar2"
  }]
});
```

An example of reading an array of DT values:

```
PLC.readArrayOfDt({  
    name: ".DateArray",  
    format: "#DD#.#MM#.#YY#, #hh#:#mm",  
    jvar: "myArray"  
});
```

The write requests for these types require a date object as input value, only or TOD strings are also accepted. Formatting parameters are not supported.

```
var myVar = new Date("January 3, 2012 18:54:00");  
  
PLC.writeDt({  
    name: ".DateVar",  
    val: myVar  
});
```

```
PLC.writeTod({  
    name: ".TODVar",  
    val: „12:33“  
});
```

Formatting Instructions

Instruction	Return Value
none	JavaScript date object
YYYY	full year
YY	year
MONTH	full name of month
MON	short name of month
MM	month 01-12, 2 digits
M	month 1-12, 1 or 2 digits
WEEKDAY	weekday, full name
WKD	weekday, short name
WD	weekday 0-6
DD	day of month 01-31, 2 digits
D	day of month 1-31, 1 or 2 digits
hh	hours, if the value is < 10 a leading zero will be added.
h	hours
mm	minutes, if the value is < 10 a leading zero will be added.
m	minutes
ss	seconds, if the value is < 10 a leading zero will be added.
s	seconds
msmsms	milliseconds, up to 2 leading zeros will be added
ms	milliseconds

12. Debugging

- If you encounter any problems you should have a look at the JavaScript console of your browser. The library generates various messages if something goes wrong. Search for the term „TAME library error:“ and read the text after it. Maybe you will get a hint what kind of problem occurred. In many cases the following lines will contain additional information like values or objects.
- You can also use the „debug“ parameter that every request supports. If you execute a request with this option set to „true“ all internal used information (descriptor, ADS parameter) for this request will be written to the console.
- During the development of a project you should use the TAME version with comments, not the minified one as error messages can be crippled.
- Don't forget that JavaScript is case sensitive.
- If you change the PLC program while having your visualisation running you must reload the browser window cause the address information of the symbols in the PLC may have changed. Even better is to close all clients before updating the PLC program.

13. Tips & Tricks

- For a good performance it is important that not too many requests are executed at the same time. I recommend to have only one request for fast cyclic polling ($\leq 1s$).
- Updating the DOM of the HTML page is far more resource intensive than fetching and parsing values. To get an idea: Parsing 2000 variables (DT and INT) takes about 20 ms with a Core-i5 PC and Firefox 10, an older PIII 2,4 GHz PC needs 50 ms. So you can read a large amount of variables with a cyclic request and update only the visible elements.
- If you encounter stuckings with fast cyclic polling on slow connections you can set an ID for the request and look if it gets better.
- Other than x86-based systems (i.e. PC's, CX10xx) ARM-based devices (i.e. CX90xx) use a 4-byte memory alignment. Every variable of 4-byte length has to be set to an address that must be divisible by 4 (compiling the project TwinCAT otherwise generates an error message).

When defining data structures the 4-byte variables should come first, then the 2-byte variables and at least the ones of 1 byte length. But you don't need to worry about that if you set the parameter „dataAlign4“ to „true“ (see Chapter 2). In this case TAME will generate padding bytes to match the alignment of the structure in the PLC's memory.

- If you need a list of the PLC variables you can use the „logSymbols()“ method. The internal symbol table will be written to the JavaScript console of your browser. The names of global PLC variables start with a point, then name of local PLC variables with the name of the instance they are defined in. They are case insensitive, „Main.IntVar“ is equal to „MAIN.INTVAR“ (this applies to all requests).
- The domain of the URL of your browser based visualisation and the one of the WebService's URL set in the parameter „serviceUrl“ of the WebServiceClient (see Chapter 2) have to be the same. Web browsers have built in a thing called „same-origin-policy“ which prevents the access to content from other domains. If you want to access the WebService across multiple domains (i.e. a from a LAN and also from the outside over DynDNS) you normally have to set 2 or more client definitions. To circumvent this you can use the property „location“ of the JavaScript window object.

So instead of using a fixed domain:

```
„serviceUrl: 'http://192.168.1.2/TcAdsWebService/TcAdsWebService.dll',“,
```

better use „window.location“:

```
„serviceUrl: window.location.protocol + '//' + window.location.hostname +  
  '/TcAdsWebService/TcAdsWebService.dll',“,
```

or, if you use only Webkit based browsers:

```
„serviceUrl: window.location.origin + '/TcAdsWebService/TcAdsWebService.dll',“.
```