



# Triamec Drive File System

## Application Note 124

A description of the file system of Triamec drives and how it is accessed. This allows reading and writing large tables, i.e. for compensation data and accessing log files.

The file system is available in firmware 4.11 and newer and is accessible with a *TAM System Explorer* version 7.15 or newer, or with a browser.

### Table of Contents

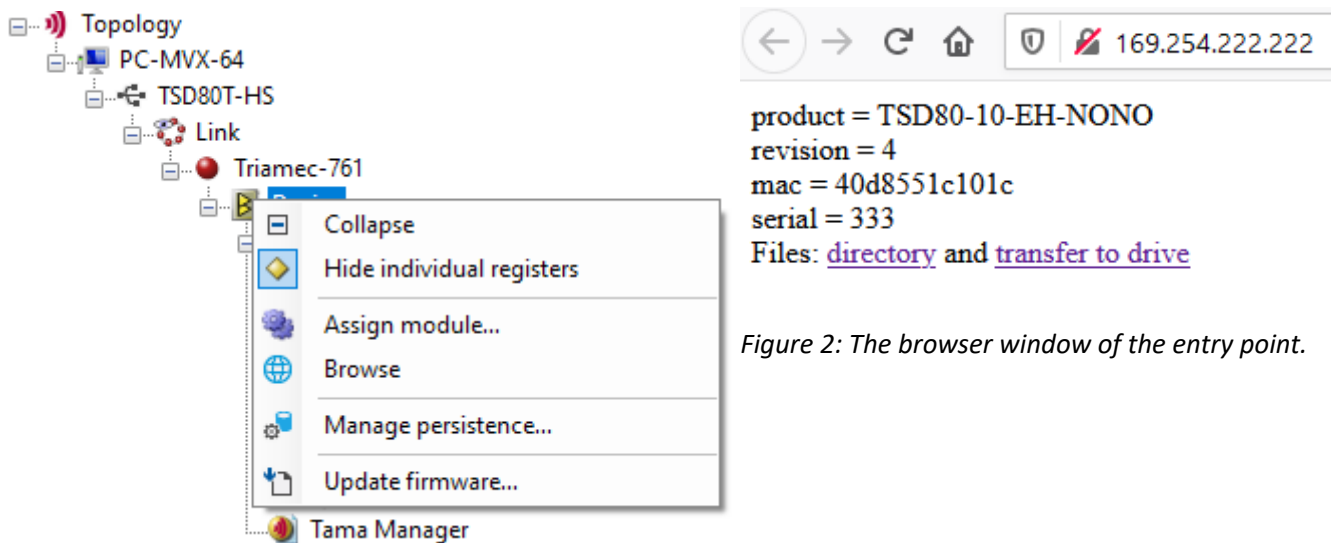
1	File System.....	2	2.4	Structure.....	6
1.1	Directory.....	3	2.5	Header.....	7
1.2	Transfer Files to Drive.....	3	2.6	Checksum Calculation.....	7
2	Tables.....	4	3	Software access.....	8
2.1	TAM System Explorer Access.....	4	3.1	Reading a file from the drive.....	8
2.2	Tama Code Access.....	5	3.2	Writing a file to the drive.....	9
2.3	File Server Access.....	5		Revision History.....	10

Document AN124\_FileSystem\_EP  
Version 009, 2024-06-13  
Source Q:\doc\ApplicationNotes\  
Destination T:\doc\ApplicationNotes  
Owner mx

# 1 File System

The entry point to the file system is the web server of the drive. For that, a valid connection to the drive has to be set up (as explained in the *Servo Drive Setup Guide*). Be aware that different connection types might not have the same performance (round trip time, etc.).

The most intuitive way of accessing the file system is using the *TAM System Explorer*. Use the context menu of the drive node as shown in Figure 1. Choose the menu item **Browse** and a browser window will open as in Figure 2. This is the entry point of the drive web server and file system access (HTTP access).

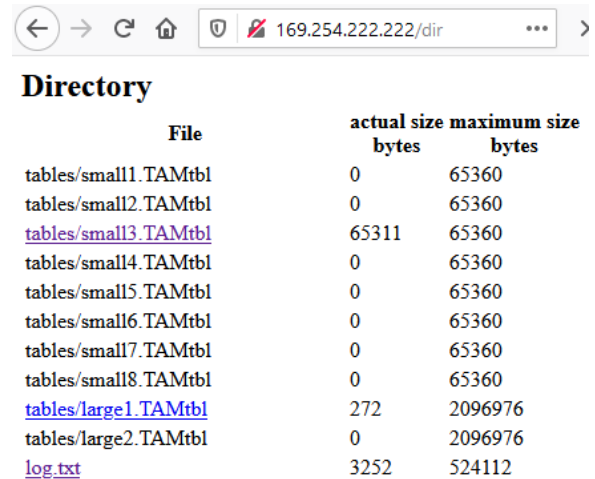


## 1.1 Directory

Open the **directory** using the link and the browser page Figure 3 appears.

The left column shows all the files, the drive knows. The second column shows the actual size in bytes. The third column is the maximum allowed size of each file.

- If an entry is marked as a link, the file contains data and may be loaded from the drive to the PC by clicking on its link.
- If the entry is plain text (without link), the file is empty and the entry is used as an indication of the maximum size of the file.



File	actual size bytes	maximum size bytes
tables/small1.TAMtbl	0	65360
tables/small2.TAMtbl	0	65360
<a href="#">tables/small3.TAMtbl</a>	65311	65360
tables/small4.TAMtbl	0	65360
tables/small5.TAMtbl	0	65360
tables/small6.TAMtbl	0	65360
tables/small7.TAMtbl	0	65360
tables/small8.TAMtbl	0	65360
<a href="#">tables/large1.TAMtbl</a>	272	2096976
tables/large2.TAMtbl	0	2096976
<a href="#">log.txt</a>	3252	524112

Figure 3: The directory page of the file system.

## 1.2 Transfer Files to Drive

Choose the **transfer to the drive** link in chapter 1 and the browser page Figure 4 opens.

- In the entry **Filepath in drive** enter a path and file name as available in the directory (see Figure 3).
- In the entry **Select from PC** choose the file on your PC that you want to transfer to the drive.
- Then choose **Start** to start transmitting.

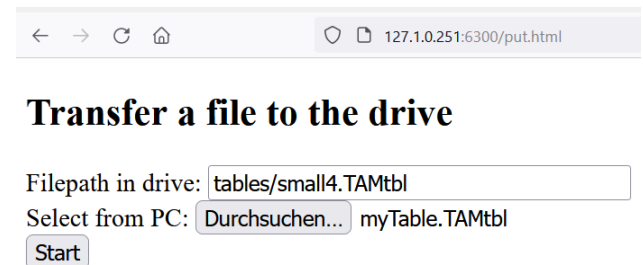


Figure 4: The transfer-a-file page

Once the browser responses with **upload of tables/small4.TAMtbl succeeded**, the file has been saved successfully to the drive ram and is accessible from Tama code.

**Warning:** If a file is a persistent file (see chapter 2), the internal saving to the permanent memory is not finished at this time. This process starts immediately after the browser finished transmitting and may take up to one second. You can work with the drive as usual and upload other files, but you should not power down the drive during this phase.

## 2 Tables

Tables can be used in a user real-time application (*Tama*), i.e. for cogging compensation. Currently we support 8 small tables (~16'000 entries) and 2 large tables (~2'000'000 entries)<sup>2</sup>.

A table contains a header and data. These may be accessed from the *TAM System Explorer* (chapter 2.1) or a user real time application in *Tama* (chapter 2.2) or from the file server (chapter 2.3).

The structure and size of a table (chapter 2.4) is specified with the header as discussed in chapter 2.5 and must be committed. This header also specifies, whether the table is persistent or temporary.

### 2.1 TAM System Explorer Access

Reading table values is done using `Application.Tables.General.Data`

- Source select the table to be accessed
- Index select the index of the table array
- Integer shows the 32bit integer value of the table at the index chosen
- Float shows the 32bit float value of the table at the index chosen
- Double shows the double value of the table at the index chosen

The value cannot be modified. Use *Tama* or the file system to set the values of the table.

Please note that Double requires two 32bit entries of the table. This means a double value at index  $i$  occupies float or integer entries at indices  $2*i$  and  $2*i+1$ .

To access the header or commands for table "Small1" for example, use register `Application.Tables.Small1`. This contains the following elements.

#### Command

The command register allows to run one of the following table commands.

Command	Description
Commit	This command is described in detail below.
Reload	Reload a persistent table from the persistent memory.
Erase	Erase the persistent memory of this table and set values to default.

Committing a table calculates the size of the table from the header parameters (see chapter Structure below). After this the table and its header can be read from the file system. If the file is persistent (see chapter 2.5) the header and data are saved to the persistent memory.

**Warning:** Committing a persistent table can wear out the persistent memory. If a certain limit has been reached, committing a table is denied with an error message and the user must wait some time before trying again.

Changing the table header without committing does not change the size of the table as visible from the file system nor the header seen from the file system nor change persistent memory. Repeat commit af-

<sup>2</sup> With firmware version < 4.2 max number of entries is ~500'000 float values.



ter changing the header and the new header will be visible from the file system and in persistent memory.

Changing the table data after committing the table changes the temporary memory (RAM) and will be immediately visible over the file system but does not update persistent memory (FLASH).

## Header

The table header is described in detail in chapter 2.5.

## State

This shows the current state of the table. 3 means the table is ready.

## 2.2 Tama Code Access

From Tama a table is accessed the same as using the *TAM System Explorer* with two exceptions:

First: Each table contains a folder “data”, which contains arrays of float, integer and double, which can directly be accessed with their indices. To set the float value of table Small1 at index 10000, for example, simply use the code:

```
Register.Application.Tables.Small1.Data.Float[10000] = 1.234f;
```

**Note:** Data.Float[100] and Data.Integer[100] and Data.Double[50] point to the same table item.

Second: To get the maximum size of a table use:

```
int len = Register.Application.Tables.Small1.Data.Float.Length;
```

To save the table persistently call the following code **once**. Don't call it frequently to prevent flash wear!

```
Register.Application.Tables.Small1.Header.Dim1.Size = size;  
Register.Application.Tables.Small1.Header.Dim2.Size = 1;  
Register.Application.Tables.Small1.Header.Dim3.Size = 1;  
Register.Application.Tables.Small1.Header.RowSize = 1;  
Register.Application.Tables.Small1.Header.Persistent = true;  
Register.Application.Tables.Small1.Command = TableCommand.Commit;
```

## 2.3 File Server Access

Tables are accessed as files using the file server as described in chapters 1.1 and 1.2.

The file starts with the table header and continues with the table data.

- The header occupies 64 words of 32bits each and is described in chapter 2.5.
- The table data consists of 32bit float or Integer entries or 64bit double entries.

The size of this file in bytes is calculated during Table Commit by

```
size = 4 * (64 + Header.RowSize * Header.Dim1.Size * Header.Dim2.Size * Header.Dim3.Size)
```

**Note:** The table is a binary file with a flat sequence of binary LittleEndian 32bit or 64bit entries without Tabulator or End-of-Line characters.

If a table is transmitted from the PC to the drive, an internal Commit of the table takes place to provide



consistency with the internal register. If the table is marked persistent (see header below) the table will then be saved to the persistent memory.

If a table is loaded from the drive to the PC, the table header is taken from the last committed version. The data part of the file is always taken from the most recent data even if the table has not been committed since changing data.

## 2.4 Structure

The structure of the table is flexible. For a standard one dimensional table of Float values, set:

- Header.RowSize = 1
- Header.Dim1.Size = number of Float values in the table
- Header.Dim2.Size = 1
- Header.Dim3.Size = 1

For a standard one dimensional table of Double values set:

- Header.RowSize = 2
- Header.Dim1.Size = number of Double values in the table
- Header.Dim2.Size = 1
- Header.Dim3.Size = 1

For a three dimensional table of Float values set:

- Header.RowSize = 1
- Header.Dim1.Size = number of entries in the first dimension
- Header.Dim2.Size = number of loops in the second dimension
- Header.Dim3.Size = number of loops in the third dimension

Generate the values of the table by something like the following code example, which sets the table to a value, that depends on the function `customerFunction(pos1, pos2, pos3)` at the three dimensional position (pos1, pos2, pos3):

```
int size1 = Register.Application.Tables.Small1.Header.Dim1.Size;
int size2 = Register.Application.Tables.Small1.Header.Dim2.Size;
int size3 = Register.Application.Tables.Small1.Header.Dim3.Size;
for (int k = 0; k < size3; k++) {           // loop over the third dimension
    for (int j = 0; j < size2; j++) {       // loop over the second dimension
        for (int i = 0; i < size1; i++) {   // loop over the first dimension
            int index = i + size1 * (j + size2 * k);
            float pos1 = Register.Application.Tables.Small1.Header.Dim1.StartValue +
                Register.Application.Tables.Small1.Header.Dim1.Distance * (float)i;
            float pos2 = Register.Application.Tables.Small1.Header.Dim2.StartValue +
                Register.Application.Tables.Small1.Header.Dim2.Distance * (float)j;
            float pos3 = Register.Application.Tables.Small1.Header.Dim3.StartValue +
                Register.Application.Tables.Small1.Header.Dim3.Distance * (float)k;
            Register.Application.Tables.Small1.Data.Float[index] = customerFunction(pos1, pos2, pos3);
        }
    }
}
```

## 2.5 Header

The header fields are shown in the following table.

Word number	Type	Register name	Description
0	Bool	Persistent	0=Table is Volatile, 1=Table is Persistent
1	Integer32	-	Must be 0
2	Integer32	Type	{0=User, 5=CoggingCompensationV1, 10=AxisCompensationV1}
3	Integer32	ChecksumMode	{0=Ignore, 1=Check, 2=Calculate}, see chapter 2.6
4-15	Integer32	Checksum	The SHA-3-384 checksum with NIST padding, set zero before calculation.
16-17	Integer64	Date	The date in 64 bit POSIX format
18	Integer32	-	Must be 0
19	Integer32	Id	A table ID given by the user
20-35	String	Description	A description string given by the user
36	Integer32	RowSize	The number of words in a row
37-39	Integer32	-	Must be 0
40	Integer32	Dim1.Size	The size of the table in the first dimension
41	Integer32	-	Must be zero
42	Float32	Dim1.StartValue	The position of the first data point of this dimension
43	Float32	Dim1.Distance	The distance between data points in this dimension
44	Integer32	Dim2.Size	The size of the table in the first dimension
45	Integer32	-	Must be zero
46	Float32	Dim2.StartValue	The position of the first data point of this dimension
47	Float32	Dim2.Distance	The distance between data points in this dimension
48	Integer32	Dim3.Size	The size of the table in the first dimension
49	Integer32	-	Must be zero
50	Float32	Dim3.StartValue	The position of the first data point of this dimension
51	Float32	Dim3.Distance	The distance between data points in this dimension
52-63	Integer32	-	Must be 0

## 2.6 Checksum Calculation

A checksum may be attached to the header. This checksum is tested in the drive if *ChecksumMode* is set to *Check*. If a file is transmitted to the drive with *ChecksumMode = Calculate*, the drive will change the *ChecksumMode* to *Check* and then calculate the checksum itself. This is useful if the user does not want to calculate the checksum himself. By reading back this file, he gets a file with a checksum value, and the *ChecksumMode = Check*.

The checksum is calculated with the SHA-3-384 method. Before calculation, zero the checksum in the header, add NIST-type of padding, then calculate the SHA3-384 hash of the file. Finally write it back into the header.



### 3 Software access

External software can access the filesystem using the IP-address shown in chapter 1. This is especially useful if the PC is connected with the drive over its auxiliary Ethernet port. With a TCP connection to port 80, external software can read and write files discussed in this document. The following code snippets show how to read and write a file.

All possible files are listed using “GET /dir”.

#### 3.1 Reading a file from the drive

Reading is done with a standard HTTP GET from the drive. To read the table “tables/small1.TAMtbl” use “GET /tables/small1.TAMtbl”. The following samples read the table “small1” from the drive, if the drive is connected using its AUTO-IP address 169.254.222.222.

##### using C#

```
var FilePath = new Uri("tables/small1.TAMtbl", UriKind.Relative); // URL to file
var baseAddress = new Uri("http://169.254.222.222/"); // drive URL
var client = new HttpClient { BaseAddress = baseAddress };
var response = new HttpResponseMessage(HttpStatusCode.Forbidden);
using (var request = new HttpRequestMessage() {
    RequestUri = FilePath,
    Method = HttpMethod.Get,
}) {
    if (client.BaseAddress != null) {
        response = await client.SendAsync(request).ConfigureAwait(false);
    }
}

if (response.StatusCode == HttpStatusCode.OK) {
    using (var fileStream = await response.Content.ReadAsStreamAsync().ConfigureAwait(false)) {
        // ...add deserialize code here
    }
}
```

##### using JavaScript

```
const fs = require('fs');
const axios = require('axios');
const FormData = require('form-data');

async function uploadFile(filePath, driveIP) {
    try {
        const data = await fs.promises.readFile(filePath);
        const formData = new FormData();
        formData.append('filename', 'tables/small1.TAMtbl');
        formData.append('filepath', data, { filename: 'anyname.bin' });
        const response = await axios.post('http://' + driveIP + '/put.html', formData, {
            headers: {
```



```
        ...formData.getHeaders(),
        'Content-Length': formData.getLengthSync()
    }
    });
    console.log('Response:', response.data);
} catch (error) {
    console.error('Error uploading file:', error);
}
}

uploadFile('dataRead.bin', '169.254.222.222');
```

### 3.2 Writing a file to the drive

Files are written using HTTP POST with a specially formatted MultipartForm object. It must contain a string component "filename" and a stream component "filepath". The following samples write a file "sample.bin" to the drive table "small1", if the drive is connected using its AUTO-IP address 169.254.222.222.

#### using C#

```
var targetName = "tables/small1.TAMtbl";
var source = new FileStream("sample.bin", FileMode.Open); // stream of sample.bin to be transferred

var uploadSite = new Uri("put.html", UriKind.Relative); // URL for file transfer
var baseAddress = new Uri("http://169.254.222.222/"); // drive URL
var client = new HttpClient { BaseAddress = baseAddress };

using (var content = new MultipartFormDataContent {
    { new StringContent(targetName), "filename" },
    { new StreamContent(source), "filepath" }
})
using (var response = await client.PostAsync(uploadSite, content)
    .ConfigureAwait(continueOnCapturedContext: false))
{
    if (!response.IsSuccessStatusCode)
    {
        throw new HttpRequestException($"Transfer to device at {targetName} failed.
{response.ReasonPhrase}");
    }
}
}
```

#### using JavaScript

```
// Create a Blob from the binaryData ArrayBuffer
const blob = new Blob([binaryData], { type: 'application/octet-stream' });
// Create a FormData object and append the Blob to it
let formData = new FormData();
const tableSlot = 'tables/small1.TAMtbl';
formData.append('filename', tableSlot);
```



```
formData.append('filepath', blob);  
// Use the Fetch API to send the FormData to the specified URL  
const response = await fetch('http://169.254.222.222/put.html', {  
  method: 'POST',  
  body: formData,  
});  
if (response.ok) {  
  console.log('Binary data sent successfully.');}  
else {  
  throw new Error(`Failed to send binary data. Status: ${response.status}`);  
}
```

## Revision History

Version	Date	Editor	Comment
001	2021-04-26	mvx	First release
002	2021-09-07	dg	renamed ColumnSize to RowSize
003	2023-03-02	sm	update template, fix header index 18 type, minor wording changes
004	2023-05-30	sm	Introduce Table Type for firmware contained features.
005	2023-08-16	sm	Fix index of Table Type.
006	2023-11-16	mvx	New interface for reading data of a table in firmware 4.20
007	2024-02-07	dg	Fix table path: /table/small1.TAMtbl → /tables/small1.TAMtbl
008	2024-03-11	ns	Fix JavaScript spelling
009	2024-04-11	dg	Size of /table/large1.TAMtbl increased to 2'000'000



---

Copyright © 2024  
Triamec Motion AG  
All rights reserved.

Triamec Motion AG  
Lindenstrasse 16  
6340 Baar / Switzerland

Phone +41 41 747 4040  
Email [info@triamec.com](mailto:info@triamec.com)  
Web [www.triamec.com](http://www.triamec.com)

## Disclaimer

This document is delivered subject to the following conditions and restrictions:

- This document contains proprietary information belonging to Triamec Motion AG. Such information is supplied solely for the purpose of assisting users of Triamec products.
- The text and graphics included in this manual are for the purpose of illustration and reference only. The specifications on which they are based are subject to change without notice.
- Information in this document is subject to change without notice.