TEMES Import Library

Table of Contents

Part I TEMES Import Library 4

1 General 4

2 Tellert Import Library 4

General 4

Data Types 4

Data Types Overview 4

Data Type: OBJECT 5

Data Type: PROPERTY 6

Data Type: COLLECTION 7

Data Type: VECTOR 8

Data Type: ITERATOR 8

Data Type: STRING 8

Data Type: TIME64 10

How to use a Tellert Import Library 11

Creating a TIL object instance 11

Connecting a TIL object instance to an import file 11

Traversing the data of a TIL object instance 12

Destroying a TIL object instance 12

3 TEMES Import Library 12

TEMES Import Object 12

TEMES Data Types 15

4 til_tms.dll 16

TIL_TMS.DLL 16

Redistribution 21

TIL_TMS.DLL 21

ZLIBWAPI.DLL 21

Supported Windows Versions 22

Version History 22

1 TEMES Import Library

1.1 General

A *TEMES Import Library* is a special kind of a *Tellert Import Library*. The first part of this document describes an excerpt of *Tellert Import Libraries* in general, and the second part describes the application of *Tellert Import Libraries* for importing TEMES files.

1.2 Tellert Import Library

1.2.1 General

A TIL (= *Tellert Import Library*) is used to access the data of files whose internal structure is unknown. The required steps of using a TIL are sketched as follows:

1. Create a void instance of an import object
2. Connect this instance with the import file
3. Traverse the instance's properties to get the data of the import file
4. Destroy the instance

1.2.2 Data Types

1.2.2.1 Data Types Overview

A TIL uses its own set of data types. Most of them are directly taken from the C-language and Windows32 environment.

Name	Abbreviation	C data type	Comment
TIL_POBJECT	Obj	void *	Pointer to an object
TIL_PPROP	Property	void *	Pointer to a property
TIL_PSTR	Str	char *	Pointer to a NUL terminated char string
TIL_PWSTR	WStr	~~unsigned short~~ * wchar_t *	Pointer to a NUL terminated wide string
TIL_DATASIZE	DataSize	~~int~~ size_t	
TIL_STRLEN	Strlen	~~int~~ size_t	Number of string characters (excluding the terminating NUL)
TIL_PROPID	PropertyID	int	
TIL_TYPEID	TypeID	int	
TIL_BOOL	Bool	~~bool~~ int	
TIL_INT8	Int8	~~char~~ __int8	
TIL_INT16	Int16	~~short~~ __int16	
TIL_INT32	Int32	~~long~~ __int32	
TIL_INT64	Int64	__int64	
TIL_UINT8	Uint8	~~unsigned char~~ unsigned __int8	
TIL_UINT16	Uint16	~~unsigned short~~ unsigned __int16	
TIL_UINT32	Uint32	~~unsigned long~~ unsigned __int32	
TIL_UINT64	Uint64	unsigned __int64	
TIL_FLOAT32	Float32	float	
TIL_FLOAT64	Float64	double	
TIL_FLOAT80	Float80	long double	
TIL_TIME64	Time64	double	Number of days since 1899-12-30 00:00:00
TIL_INT		int	
TIL_UINT		unsigned int	
TIL_SIZET		size_t	
TIL_CHAR		char	
TIL_WCHAR		wchar_t	
TIL_TCHAR		TCHAR	Either char or wchar_t
TIL_PTSTR		LPTSTR	Either char * or wchar_t *

Pointers of a data type are prefixed with *TIL_P* or *TIL_LP*. Pointers to constant objects are prefixed with *TIL_PC* or *TIL_LPC*.

1.2.2.2 Data Type: OBJECT

An object instance is represented by a non-null pointer of type *TIL_POBJECT*. If a function, which returns a *TIL_POBJECT,* is called, the *TilDestroyObject* function with the object instance pointer as argument must be called as soon as the object instance is no longer needed.

Following functions are available to create and destroy objects:

```
TIL_POBJECT TilCreateObject(TIL_PCTSTR pObjectGUID);
TIL_POBJECT TilCreateObjectA(TIL_PCSTR pObjectGUID);
TIL_POBJECT TilCreateObjectW(TIL_PCWSTR pObjectGUID);

void TilDestroyObject(TIL_POBJECT pObject);
```

1.2.2.3 Data Type: PROPERTY

Properties exist within the scope and life time of an object instance. They represent either a basic data type (like *unsigned*), a complex data type (like *string*) or a set of further properties (*container* or *vector*). All properties of an object are contained within the root property of an object. The root property is obtained by calling

```
TIL_PPROP TilGetRootProperty(TIL_POBJECT pObject);
```

The internal type of a property can be obtained by calling

```
TIL_TYPEID TilGetType(TIL_PCPROP pProp);
```

where the returned type ID belongs to the corresponding data type as follows:

Property data type	**Declaration**	**Property data type ID (TYPEID)**
VOID		TILID_VOID (= 0)
COLLECTION	*name* { … };	TILID_COLLECTION (= 128)
VECTOR	…*name*[]…;	TILID_VECTOR (= 129)
ITERATOR	iterator *name*;	TILID_ITERATOR (= 130)
TIL_PDATA	data *name*;	TILID_DATA (= 64)
TIL_PSTR	string *name*;	TILID_STR (= 65)
TIL_PWSTR	string *name*;	TILID_WSTR (= 66)
TIL_BOOL	bool *name*;	TILID_BOOL (= 1)
TIL_INT8	int8 *name*;	TILID_INT8 (= 2)
TIL_INT16	int16 *name*;	TILID_INT16 (= 3)
TIL_INT32	int32 *name*;	TILID_INT32 (= 4)
TIL_INT64	int64 *name*;	TILID_INT64 (= 5)
TIL_UINT8	uint8 *name*;	TILID_UINT8 (= 8)
TIL_UINT16	uint16 *name*;	TILID_UINT16 (= 9)
TIL_UINT32	uint32 *name*;	TILID_UINT32 (= 10)
TIL_UINT64	uint64 *name*;	TILID_UINT64 (= 11)
TIL_FLOAT32	float32 *name*;	TILID_FLOAT32 (= 32)
TIL_FLOAT64	float64 *name*;	TILID_FLOAT64 (= 33)
TIL_FLOAT80	float80 *name*;	TILID_FLOAT80 (= 34)
TIL_TIME64	time64 *name*;	TILID_TIME64 (= 40)
TIL_INT		TILID_INT
TIL_UINT		TILID_UINT
TIL_SIZET		TILID_SIZET

The following functions can be used to get the value of a property for basic data types:

```
TIL_BOOL TilGetBool(TIL_PCPROP pProperty, TIL_PBOOL pBool);
TIL_BOOL TilGetInt8(TIL_PCPROP pProperty, TIL_PINT8 pInt8);
```

```
TIL_BOOL TilGetInt16(TIL_PCPROP pProperty, TIL_PINT16 pInt16);
TIL_BOOL TilGetInt32(TIL_PCPROP pProperty, TIL_PINT32 pInt32);
TIL_BOOL TilGetInt64(TIL_PCPROP pProperty, TIL_PINT64 pInt64);
TIL_BOOL TilGetUInt8(TIL_PCPROP pProperty, TIL_PUINT8 pUInt8);
TIL_BOOL TilGetUInt16(TIL_PCPROP pProperty, TIL_PUINT16 pUInt16);
TIL_BOOL TilGetUInt32(TIL_PCPROP pProperty, TIL_PUINT32 pUInt32);
TIL_BOOL TilGetUInt64(TIL_PCPROP pProperty, TIL_PUINT64 pUInt64);
TIL_BOOL TilGetFloat32(TIL_PCPROP pProperty, TIL_PFLOAT32 pFloat32);
TIL_BOOL TilGetFloat64(TIL_PCPROP pProperty, TIL_PFLOAT64 pFloat64);
TIL_BOOL TilGetFloat80(TIL_PCPROP pProperty, TIL_PFLOAT80 pFloat80);
TIL_BOOL TilGetTime64(TIL_PCPROP pProperty, TIL_PTIME64 pTime64);
TIL_BOOL TilGetInt(TIL_PCPROP pProperty, TIL_PINT pInt);
TIL_BOOL TilGetUInt(TIL_PCPROP pProperty, TIL_PUINT pUInt);
TIL_BOOL TilGetSizeT(TIL_PCPROP pProperty, TIL_PSIZET pSizeT);
```

Accessing *Collections*, *Vectors*, *Iterators* and *Strings* are described separately.

Note, that the data type *TIL_FLOAT80* is not supported by all compilers. E.g. the 64-bit-version of *TilGetFloat80* always returns false.

1.2.2.4 Data Type: COLLECTION

Collections are properties which represent a set of properties. The functions

```
TIL_PPROP TilGetProperty(TIL_PPROP pProp, TIL_PROPID PropID);
TIL_PPROP TilGetPropertyEx(TIL_PPROP pProp, TIL_PROPID PropID,
 TIL_PTYPEID pTypeID);
```

are used to access the properties of a collection, where *PropID* is the desired property tag and *pTypeID* is a destination for the type of the corresponding property.

A collection is declared by its name followed by an opening brace, a description of the collection items, a closing brace and a semicolon. Collection items are described according to their declaration (which is for non-set data types: data type followed by item name followed by semicolon).

The data type *variant* is used for items which may be of any type.

The *PropID* value of a collection item is optionally written after the item name or after any name decorations.

Example:

```
Collection {
    string CollectionItem1 = 1;
    SubCollection = 2 {
        string SubCollectionItem1 = 1;
        string SubCollectionItem2 = 2;
    };
};
```

1.2.2.5 Data Type: VECTOR

Vectors are properties which represent a one dimensional vector of properties. The following functions can be used to access a vector:

```
TIL_DATASIZE TilGetVectorSize(TIL_PCPROP pVector);
TIL_PPROP TilGetVectorItem(TIL_PCPROP pVector, TIL_DATASIZE Position);
TIL_PPROP TilGetVectorItemEx(TIL_PCPROP pVector, TIL_DATASIZE Positi-
 on, TIL_PTYPEID pTypeID);
```

A vector is declared by decorating the corresponding property name with a postfixed opening bracket followed by a closing bracket.

Example:

```
Collection {
    CollectionVector[] = 1 {
        uint16 UnsignedVector[] = 2;
    };
};
```

1.2.2.6 Data Type: ITERATOR

Iterators are properties which are used to traverse vectorized data. Following functions can be used to extract data with iterators:

```
TIL_DATASIZE TilGetIteratorItemSize(TIL_PCPROP pIterator);
TIL_DATASIZE TilGetIteratorItemCount(TIL_PCPROP pIterator);

TIL_BOOL TilRewindIterator(TIL_PPROP pIterator);
TIL_BOOL TilGetNextIteratorItems(TIL_PPROP pIterator, void *pDest,
 TIL_DATASIZE nCount);
```

Example of extracting data:

```
size_t ItemSize = TilGetIteratoItemSize(pIterator);
size_t ItemCount = TilGetIteratorItemCount(pIterator);
char *data[] = new char [ItemSize*ItemCount];
TilRewindIterator(pIterator);
TilGetNextIteratorItems(pIterator, data, ItemCount);
delete[] data;
```

1.2.2.7 Data Type: STRING

Strings are properties which represent NUL-terminated arrays of type *char* or of type *wchar_t*. Functions belonging to strings of type *char* are postfixed with letter *A*, and functions belonging to strings of type *wchar_t* are postfixed with letter *W*. The following functions can be used to read strings:

```
TIL_BOOL TilGetString(TIL_PPROP pStrProp, TIL_PTSTR pStr, TIL_STRLEN
 MaxStringLen, TIL_PSTRLEN StringLen);
TIL_BOOL TilGetStringA(TIL_PPROP pStrProp, TIL_PSTR pStr, TIL_STRLEN
```

```
 MaxStringLen, TIL_PSTRLEN StringLen);
TIL_BOOL TilGetStringW(TIL_PPROP pStrProp, TIL_PWSTR pStr, TIL_STRLEN
 MaxStringLen, TIL_PSTRLEN StringLen);
```

The following functions can be used to modify strings:

```
TIL_BOOL TilSetString(TIL_PPROP pStrProp, TIL_PCTSTR pStr);
TIL_BOOL TilSetStringA(TIL_PPROP pStrProp, TIL_PCSTR pStr);
TIL_BOOL TilSetStringW(TIL_PPROP pStrProp, TIL_PCWSTR pStr);
TIL_BOOL TilSetStringEx(TIL_PPROP pStrProp, TIL_PCTSTR pStr,
 TIL_STRLEN StrLen);
TIL_BOOL TilSetStringExA(TIL_PPROP pStrProp, TIL_PCSTR pStr,
 TIL_STRLEN StrLen);
TIL_BOOL TilSetStringExW(TIL_PPROP pStrProp, TIL_PCWSTR pStr,
 TIL_STRLEN StrLen);
```

The following macro can be used to declare a TIL_PCTSTR literal:

```
TIL_TEXT("string") or TIL_T("string")
```

Example:

```
#define TIL_QUERY_STRLEN ((TIL_STRLEN)-1)

TIL_PCTSTR InvalidString = TIL_TEXT("");

TIL_PCTSTR NewString(TIL_PPROP prop)
{
    TIL_STRLEN len;

    /* Initialize len, because TilGetString will fail due to an inva-
 lid buffer */
    len = TIL_QUERY_STRLEN;
    TilGetString(prop, NULL, TIL_QUERY_STRLEN, &len);
    if (len != TIL_QUERY_STRLEN) {
        /* allocate one additional character for the terminating NUL
 */
        TIL_PTSTR str = (TIL_PTSTR) malloc( (len+1) * si-
 zeof(TIL_TCHAR) );
        if (str) {
            if (TilGetString(prop, str, len, NULL)) {
                return str;
            }
            free(str);
        }
    }

    return InvalidString;
}

void DeleteString(TIL_PCTSTR str)
{
    if (str != InvalidString) {
        free((void*)str);
    }
}
```

1.2.2.8 Data Type: TIME64

Properties of type TIME64 represent the number of days since 1899-12-30 00:00:00. If you cannot handle the floating point value (number of days), you can query the STRING version of the floating point value (*TilGetString*). This STRING version is build as "YYYY-MM-DD hh:mm:ss.ffffff" with

Y: Year digit
M: Month digit
D: Day digit
h: Hour digit (hh = 0…23)
m: Minute digit (mm = 0…59)
s: Second digit (ss = 0…59)
f: Fractional part of one second

Example:

```
#include <windows.h>
#include "til.h"

TIL_BOOL TilTime64ToTilDateTime(TIL_TIME64 t, TIL_PDATETIME pdt)
{
    SYSTEMTIME    st;
    FILETIME      ft;
    LARGE_INTEGER li;

    if (pdt == NULL) {
        return TIL_FALSE;
    }

    memset(pdt, '\0', sizeof(TIL_DATETIME));
    if (t < -109205 || t > 10565994) {
        return TIL_FALSE;
    }
    if (t == 0) {
        return TIL_TRUE;
    }

    li.QuadPart  = (LONGLONG) (t * 864000000000i64);
    li.QuadPart += 94353120000000005i64;
    ft.dwLowDateTime  = li.LowPart;
    ft.dwHighDateTime = li.HighPart;
    if (FileTimeToSystemTime(&ft, &st)) {
        pdt->year  = (TIL_UINT16) st.wYear;
        pdt->month = (TIL_UINT16) st.wMonth;
        pdt->day   = (TIL_UINT16) st.wDay;
        pdt->hour  = (TIL_UINT16) st.wHour;
        pdt->min   = (TIL_UINT16) st.wMinute;
        pdt->sec   = (TIL_UINT16) st.wSecond;
        pdt->usec  = (TIL_UINT32) (li.QuadPart % 10000000) / 10;
    }

    return TIL_TRUE;
```

```
}
```

1.2.3 How to use a Tellert Import Library

1.2.3.1 Creating a TIL object instance

A TIL object instance is created by the functions

```
TIL_POBJECT TilCreateObject(TIL_PCTSTR pObjectGUID);
TIL_POBJECT TilCreateObjectA(TIL_PCSTR pcObjectGUID);
TIL_POBJECT TilCreateObjectW(TIL_PCWSTR pcObjectGUID);
```

where *pObjectGUID* is the identification string of the import object. The return value is either a pointer to the import object instance or *null* if the function fails.

A call of function

```
TIL_PPROP TilGetRootProperty(TIL_POBJECT pObject);
```

returns the root property of a TIL object instance. It is declared as

```
Root {
    Control = 3 {
        string FileName = 3;
    };
    variant Data = 4;
};
```

1.2.3.2 Connecting a TIL object instance to an import file

The instance of a TIL object is connected to an import file by setting the *Control.FileName* property. The following example creates an instance of a TIL object and connects it to file "*measurement.tms*":

```
TIL_POBJECT pTemesImportObject = TilCreateObject(TIL_TEXT("{35F0EC67-
 B7ED-4627-9EF6-563016B176B5}"));

if (pTemesImportObject != 0) {

  TIL_PPROP Root = TilGetRootProperty(pTemesImportObject);

  /* Connect import file */
  TIL_PPROP Control = TilGetProperty(Root, tilControl);
  TIL_PPROP FileName = TilGetProperty(Control, tilControlFileName);
  if (TilSetString(FileName, TIL_TEXT("measurement.tms"))) {
     /* Traverse data */
  } else {
     /* Cannot read measurement.tms */
     MessageBox(0, _TEXT("Cannot open import file!"), _TEXT("ERROR"),
      MB_OK|MB_ICONHAND);
  }

  /* Destroy TIL object */
```

```
}
```

1.2.3.3 Traversing the data of a TIL object instance

Once a TIL object instance is connected to the import file, the *Root.Data* property is not null any more and may be traversed. The declaration of *Root.Data* depends on the kind of import library. A declaration for TEMES import libraries is found in the section *TEMES Import Library.*

1.2.3.4 Destroying a TIL object instance

If the instance of a TIL object (and all of its possible temporary files) is no longer needed, then its resources must be freed with the call of

```
void TilDestroyObject(TIL_POBJECT pObject);
```

1.3 TEMES Import Library

1.3.1 TEMES Import Object

A TEMES import object is a TIL import object and declared as follows (Gray items are not supported up to now by the import library *til_tms.dll*):

```
Root {
    Local = 1 {
        uint16 LangID = 1;
        uint16 ResolvedLangID = 2;
        uint16 SupportedLangIDs[] = 3;
    };
    Info = 2 {
        SupportedTypes[] = 1 {
            string GUID = 1;
            string Description = 2;
            string Extension = 3;
            string ArgGUIDs[] = 4;
            Features[] = 5 {
                string GUID = 1;
                variant Data = 2;
            };
        };
    };
    Control = 3 {
        int16 TypeIndex = 1;
        Args[] = 2 {
            string GUID = 1;
            variant Data = 2;
        };
        string FileName = 3;
    };
    Data = 4 {
        Comm = 1 {
            string Name = 1;
            string Comment = 2;
```

```
    Variable[] = 3 {
        variant Key = 1;
        variant Value = 2;
    };
};
Meas[] = 3 {
    Comm = 1 {
        string Name = 1;
        string Comment = 2;
        Variable[] = 3 {
            variant Key = 1;
            variant Value = 2;
        };
    };
    Group[] = 2 {
        Comm = 1 {
            string Name = 1;
            string Comment = 2;
            Variable[] = 3 {
                variant Key = 1;
                variant Value = 2;
            };
        };
        uint64 SampleCount = 2;
        float64 SampleRate = 3;
        Info = 4 {
            Time = 1 {
                time64 HWSetup = 1;
                time64 Start = 2;
                time64 HWReadOut = 3;
                Stamp Section[] = 4 {
                    uint64 SampleIndex = 1;
                    time64 Time Start = 2;
                };
            };
        };
        Signal[] = 5 {
            Comm = 1 {
                string Name = 1;
                string Comment = 2;
                Variable[] = 3 {
                    variant Key = 1;
                    variant Value = 2;
                };
            };
            string InternalName = 2;
            string Unit = 3;
            Data = 4 {
                Raw = 1 {
                    uint16 Type = 1;
                    float64 Factor = 2;
                    float64 Offset = 3;
                    File = 4 {
                        string Name = 1;
                        uint64 Offset = 2;
                        uint32 DataSize = 3;
                        uint32 Gap = 4;
                    };
                    iterator Iterator = 5;
```

```
                    };
                    float64 InvalidValue = 2;
                    iterator Iterator = 3;
                    Flags = 4 {
                        bool Quantized = 1;
                        bool WrapAroundCounter = 2;
                    };
                };
                RefSignal = 5 {
                    float64 Factor = 1;
                    float64 Offset = 2;
                };
                View = 6 {
                    float64 Min = 1;
                    float64 Max = 2;
                };
                Marker[] = 7 {
                    string Name = 1;
                    uint64 SampleIndex = 2;
                    bool Visible = 3;
                    uint32 Type = 4;
                    string Text = 5;
                    uint32 TextAngle = 6;
                    uint32 TextDistance = 7;
                };
            };
        };
    };
    View[] = 4 {
        Comm = 1 {
            string Name = 1;
            string Comment = 2;
            Variable[] = 3 {
                variant Key = 1;
                variant Value = 2;
            };
        };
        Item[] = 2 {
            Comm = 1 {
                string Name = 1;
                string Comment = 2;
                Variable[] = 3 {
                    variant Key = 1;
                    variant Value = 2;
                };
            };
            SignalInfo = 2 {
                uint32 MeasIndex = 1;
                uint32 GroupIndex = 2;
                uint32 SignalIndex = 3;
            };
            string Unit = 3;
            float64 Factor = 4;
            float64 Offset = 5;
            float64 RefFactor = 6;
            float64 RefOffset =7;
            bool Visible = 8;
        };
    };
```

```
        };
    };
```

1.3.2 TEMES Data Types

The following table displays the supported types for signal raw data. Gray entries are not supported up to now, but may be used as mentioned in the future.

DataTypeName	DataTypeID	DataTypeSize in bytes	Comment
dtUnknown	0	unknown	Unknown data type
dtSampleNumber	1	0	Raw data value equals the corresponding sample number
dt_u8	2	1	unsigned char (Intel)
dt_s8	3	1	signed char (Intel)
dt_u16	4	2	unsigned short (Intel)
dt_s16	5	2	signed short (Intel)
dt_u32	6	4	unsigned long (Intel)
dt_s32	7	4	signed long (Intel)
dt_u64	8	8	unsigned __int64 (Intel)
dt_s64	9	8	signed __int64 (Intel)
dt_U8	16	1	unsigned char (Motorola); compatible to dt_u8 but indicating that the data source is designed for Motorola endian.
dt_S8	17	1	signed char (Motorola); compatible to dt_s8 but indicating that the data source is designed for Motorola endian.
dt_U16	18	2	unsigned short (Motorola)
dt_S16	19	2	signed short (Motorola)
dt_U32	20	4	unsigned long (Motorola)
dt_S32	21	4	signed long (Motorola)
dt_U64	22	8	unsigned __int 64 (Motorola)
dt_S64	23	8	signed __int64 (Motorola)
dt_f32	32	4	float (Intel)
dt_f64	33	8	double (Intel)
dt_f80	34	10	long double (Intel)
dt_F32	48	4	float (Motorola)
dt_F64	49	8	double (Motorola)
dt_F80	50	10	long double (Motorola)
dt_c8	64	1	char
dt_c16	65	2	unsigned short (Intel)
dt_t64	80	8	Number of days since 1899-12-30 00:00:00

1.4 til_tms.dll

1.4.1 TIL_TMS.DLL

The library *TIL_TMS.DLL* realizes a subset of a general TEMES Import Library. Only the properties listed below are supported:

```
Root {
    Control = 3 {
        string FileName = 3;
    };
    Data = 4 {
        Meas[] = 3 {
            Comm = 1 {
                Variable[] = 3 {
                    variant Key = 1;
                    variant Value = 2;
                };
            };
            Group[] = 2 {
                Comm = 1 {
                    string Name = 1;
                    string Comment = 2;
                    Variable[] = 3 {
                        variant Key = 1;
                        variant Value = 2;
                    };
                };
                uint64 SampleCount = 2;
                float64 SampleRate = 3;
                Info = 4 {
                    Time = 1 {
                        time64 HWSetup = 1;
                        time64 Start = 2;
                        time64 HWReadOut = 3;
                        Stamp Section[] = 4 {
                            uint64 SampleIndex = 1;
                            time64 Time Start = 2;
                        };
                    };
                };
                Signal[] = 5 {
                    Comm = 1 {
                        string Name = 1;
                        string Comment = 2;
                        Variable[] = 3 {
                            variant Key = 1;
                            variant Value = 2;
                        };
                    };
                    string Unit = 3;
                    Data = 4 {
                        Raw = 1 {
                            uint16 Type = 1;
                            float64 Factor = 2;
                            float64 Offset = 3;
                            File = 4 {
                                string Name = 1;
```

```
                                uint64 Offset = 2;
                                uint32 DataSize = 3;
                                uint32 Gap = 4;
                            };
                            iterator Iterator = 5;
                        };
                        iterator Iterator = 3;
                        Flags = 4 {
                            bool Quantized = 1;
                        };
                    };
                    View = 6 {
                        float64 Min = 1;
                        float64 Max = 2;
                    };
                    Marker[] = 7 {
                        string Name = 1;
                        uint64 SampleIndex = 2;
                        bool Visible = 3;
                        uint32 Type = 4;
                        string Text = 5;
                        uint32 TextAngle = 6;
                        uint32 TextDistance = 7;
                    };
                };
            };
        };
        View[] = 4 {
            Comm = 1 {
                string Name = 1;
            };
            Item[] = 2 {
                Comm = 1 {
                    string Name = 1;
                };
                SignalInfo = 2 {
                    uint32 MeasIndex = 1;
                    uint32 GroupIndex = 2;
                    uint32 SignalIndex = 3;
                };
                string Unit = 3;
                float64 Factor = 4;
                float64 Offset = 5;
                float64 RefOffset =7;
                bool Visible = 8;
            };
        };
    };
};
```

Control.FileName: Setting this property to a valid TEMES filename fills the *Data* property with the content of the TEMES file.

Data.Meas: *TIL_TMS.DLL* supports one measurement up to now. Thus, the vector size of this property is always 1.

Data.Meas[0].Comm.Variable[]: Root variables of a TEMES file.

Data.Meas[0].Group[]: A measurement group combines a set of signals together which have several properties in common (sample rate, sample count, reference signal, etc.)

Data.Meas[0].Group[].Comm[].Name: Name of a group (e.g. "Fast" or "Slow"). Note that group names can be empty and may not be unambiguous.

Data.Meas[0].Group[].Comm[].Comment: Group comment (which is in fact the comment of the measurement).

Data.Meas[0].Group[].Comm[].Variable[]: Group variables.

Data.Meas[0].Group[].SampleCount: Number of samples.

Data.Meas[0].Group[].SampleRate: Sample rate in seconds.

Data.Meas[0].Group[].Info.Time.HWSetup: Time when the hardware device was set up.

Data.Meas[0].Group[].Info.Time.Start: Time of the first sample of the signal group.

Data.Meas[0].Group[].Info.Time.HWReadOut: Time when the measurement has been copied from the hardware device to the PC.

Data.Meas[0].Group[].Info.Time.~~Stamp~~Section[]: List of sample sections (defined by first sample index and start time). Note that the sample indices of this list are monotonically increasing and unique. Sections are mutually exclusive, that means each section ends before the next section starts.

Data.Meas[0].Group[].Info.Time.~~Stamp~~Section[].SampleIndex: Sample index of the first section sample (starting from zero)

Data.Meas[0].Group[].Info.Time.~~Stamp~~Section[].~~Time~~Start: Time of the first section sample

Data.Meas[0].Group[].Signal[]: By definition the first signal (= DataMeas[].Group[].Signal[0]) of a measurement group is the reference signal to which all the other signals of a measurement group refer. In most cases this is the time signal.

Data.Meas[0].Group[].Signal[].Comm.Name: Signal name.

Data.Meas[0].Group[].Signal[].Comm.Comment: For future use: This property exists in a TEMES file but is empty.

Data.Meas[0].Group[].Signal[].Comm.Variable[]: For future use: This property exists in a TEMES file but is empty.

Data.Meas[0].Group[].Signal[].Unit: Physical unit (e.g. "s", "V", "kph" or "rpm").

Data.Meas[0].Group[].Signal[].Data: This property represents the signal values.

Data.Meas[0].Group[].Signal[].Data.Raw: This property represents the raw signal data (Quantized values in most cases). The signal value is received by *Raw.Factor * RawData + Raw.Offset*

Data.Meas[0].Group[].Signal[].Data.Raw.Type: This property specifies the type of the raw data according to the following table:

DataTypeName	DataTypeID	DataTypeSize in bytes	Comment
dtSampleNumber	1	0	Raw data value equals the corresponding sample number
dt_u8	2	1	unsigned char (Intel)
dt_s8	3	1	signed char (Intel)
dt_u16	4	2	unsigned short (Intel)
dt_s16	5	2	signed short (Intel)
dt_u32	6	4	unsigned long (Intel)
dt_s32	7	4	signed long (Intel)
dt_u64	8	8	unsigned __int64 (Intel)
dt_s64	9	8	signed __int64 (Intel)
dt_f32	32	4	float (Intel)
dt_f64	33	8	double (Intel)
dt_f80	34	10	long double (Intel) (Note: Not yet used but supported by TEMES)

Data.Meas[0].Group[].Signal[].Data.Raw.Factor: Multiplier for the quantized signal value.

Data.Meas[0].Group[].Signal[].Data.Raw.Offset: Offset for the quantized signal value.

Data.Meas[0].Group[].Signal[].Data.Raw.File: Location of the signal raw data if *Raw.Type* is not *dtSampleNumber*.

Data.Meas[0].Group[].Signal[].Data.Raw.File.Name: File name of signal raw data. Note, that the file accesses must be finished before*TilDestroyObject* is called.

Data.Meas[0].Group[].Signal[].Data.Raw.File.Offset: File offset (= Position of the first byte of the first raw data value).

Data.Meas[0].Group[].Signal[].Data.Raw.File.DataSize: Size of the raw data value.

Data.Meas[0].Group[].Signal[].Data.Raw.File.Gap: Number of bytes which separate two raw data values.

Data.Meas[0].Group[].Signal[].Data.Flags.Quantized: If present, this property indicates that the raw data values may not be interpolated (e.g. *register values* or *segment number information).*

Data.Meas[0].Group[].Signal[].View.Min: Physical value for the axis start.

Data.Meas[0].Group[].Signal[].View.Max: Physical value for the axis end.

Data.Meas[0].Group[].Signal[].Marker[]: List of signal markers.

Data.Meas[0].Group[].Signal[].Marker[].Name: Marker identifier.

Data.Meas[0].Group[].Signal[].Marker[].SampleIndex: Marker sample index. Note that this property may be missing.

Data.Meas[0].Group[].Signal[].Marker[].Visible: Marker visibility. Note that this property may be missing.

Data.Meas[0].Group[].Signal[].Marker[].Type: Marker type (indicating the "heading level" – the lower the number the bigger the marker).

Data.Meas[0].Group[].Signal[].Marker[].Text: Visible marker text.

Data.Meas[0].Group[].Signal[].Marker[].TextAngle: Marker text angle with a resolution of 0.01 degrees per bit. The angle starts at 3 o'clock position and increases counterclockwise. Note that this property may be missing.

Data.Meas[0].Group[].Signal[].Marker[].TextDistance: Marker text distance with a resolution of 0.01 mm per bit. Note that this property may be missing.

Data.View: Vector containing the different views of a TEMES file. The first and unnamed view is the default view.

Data.View[].Comm.Name: Name of the view. This property does not exist for the first view (= default view).

Data.View[].Item[]: Vector containing the different displayed signals.

Data.View[].Item[].Comm.Name: Displayed name of the signal.

Data.View[].Item[].SignalInfo: Reference to the signal which is defined by **Data.Meas[]**

Data.View[].Item[].SignalInfo.MeasIndex: Zero-based index of the respective **Data.Meas** item. This entry is always 0 at the moment (since **Data.Meas** contains exactly one item).

Data.View[].Item[].SignalInfo.GroupIndex: Zero-based index of the respective **Data.Meas[].Group** item.

Data.View[].Item[].SignalInfo.SignalIndex: Zero-based index of the respective **Data.Meas[].Group[].Signal** item.

Data.View[].Item[].Unit: Displayed physical unit of the signal.

Data.View[].Item[].Factor: Additional multiplier for the signal.

Data.View[].Item[].Offset: Additional offset for the signal.

Data.View[].Item[].RefOffset: Additional offset for the reference signal.

The value of the displayed signal is obtained by ItemValue(*t*) = *Factor* * SignalValue(*t* - *RefOffset*) + *Offset*

Data.View[].Item[].Visible: Item visibility. Note that this property may be missing.

Note that the properties **Data.Meas[0].Group[].Signal[i].Data.Raw.File** and **Data.Meas[0].Group[].Signal[i].Data.Raw.Iterator** are not created if **Data.Meas[0].Group[].Signal[i].Data.Raw.Type** equals *dtSampleNumber*. Note, that property **Data.Meas[0].Group[].Signal[i].Data.Iterator** is not created if both the property **Data.Meas[0].Group[].Signal[i].Data.Raw.Type** equals *dt_f80* and the function *TilGetFloat80* is not supported.

1.4.2 Redistribution

1.4.2.1 TIL_TMS.DLL

All redistributable *TIL_TMS.DLL* files are located in the subfolders of folder "*redist\til_tms.dll*".

Permission is hereby granted, free of charge, to any person obtaining a copy of these *TIL_TMS.DLL* files, to deal in these *TIL_TMS.DLL* files without restriction, including without limitation on the rights to use, copy, compress, rename, embed, merge, publish, distribute, sublicense, and/or sell copies of these *TIL_TMS.DLL* files. However, further modifications of the *TIL_TMS.DLL* files, or a reverse engineering of these *TIL_TMS.DLL* files is not allowed.

THE "TIL_TMS.DLL" FILES ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.4.2.2 ZLIBWAPI.DLL

All redistributable *ZLIBWAPI.DLL* files are located in the subfolders of folder "*redist\zlibwapi.dll*".

This software (= *ZLIBWAPI.DLL*) is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

(Further information is available at http://www.winimage.com/zLibDll/minizip.html and http://zlib.net)

1.4.3 Supported Windows Versions

TIL_TMS.DLL is available in four different versions:

- Ansi: This 32-bit version supports the active code page
- Unicode: This 32-bit version supports also pure Unicode characters
- Unicode-Ansi: This 32-bit version uses the Ansi version if TIL_TMS.DLL is run on Windows Me/98/95, and the Unicode version if TIL_TMS.DLL is run on Windows 10/8.1/8/7/Vista/XP/2000/NT4.
- x64: This 64-bit version supports pure Unicode characters and Windows x64-Editions

	Unicode-Ansi (32-bit)	Ansi (32-bit)	Unicode (32-bit)	x64 (64-bit)
Windows 10 x64-Edition	X	X	X	X
Windows 10 (32-bit)	X	X	X	-
Windows 8.1 x64-Edition	X	X	X	X
Windows 8.1 (32-bit)	X	X	X	-
Windows 8 x64-Edition	X	X	X	X
Windows 8 (32-bit)	X	X	X	-
Windows 7 x64-Edition	X	X	X	X
Windows 7 (32-bit)	X	X	X	-
Windows Vista x64-Edition	X	X	X	X
Windows Vista (32-bit)	X	X	X	-
Windows XP x64-Edition	X	X	X	X
Windows XP (32-bit)	X	X	X	-
Windows 2000	X	X	X	-
Windows NT4	X	X	X	-
Windows Me	X	X	-	-
Windows 98	X	X	-	-
Windows 95	X	X	-	-

1.4.4 Version History

V1.3.2

- New: Support for platform dependent *TIL_TMS.DLL* names. When the TIL DLL name contains the letter sequence *tms*, then the corresponding *tma* DLL is searched first as a replacement for *ZLIBWAPI.DLL*. (*TIL_TMS.DLL* à *TIL_TMA.DLL*, see also *\source\c\lib\compiler.txt*)

V1.3.1

- New: Support for properties *tilDataMeasComm*, *tilDataMeasCommVariable* and all its items, *tilDataMeasGroupComm* and all its items, *tilDataMeasGroupSignalCommComment*, *tilDataMeasGroupSignalCommVariable* and all its items, *tilDataMeasGroupSignalMarker* and all its items, *tilDataView* and all its items.

V1.3.0

- New: If *TIL_TMS.DLL* finds the additional redistributable file *ZLIBWAPI.DLL* inside its storage fol-

der, then *TIL_TMS.DLL* can also open compressed TEMES files (file extension *.tma*).

V1.2.1
- New: Unicode-Ansi version. The 32-bit DLL uses the Ansi version if TIL_TMS.DLL is run on Windows Me/98/95, and the Unicode version if TIL_TMS.DLL is run on Windows 10/8.1/8/7/Vista/ XP/2000/NT4.

V1.2.0
- New: Support for x64 environments. (However, functions of the x64 version with data type *TIL_FLOAT80* always return false. These are the functions *TilGetFloat80*, *TilSetFloat80* and *TilGetNextIteratorItems* for *tilDataMeasGroupSignalDataIterator* with data type *TILID_FLOAT80*).
- New: Additional and redundant type definitions (to simplify TIL API usage).
- Changed: Stricter type definitions (to support 64-bit environments).
- Changed: Changed property names from *Stamp* to *Section* and from *Stamp.Time* to *Section.Start* for a better clarity of the TEMES timestamp concept. However, the obsolete naming convention is still supported.
  tilDataMeasGroupInfoTimeStamp → *tilDataMeasGroupInfoTimeSection*
  tilDataMeasGroupInfoTimeStampSampleIndex → *tilDataMeasGroupInfoTimeSectionSampleIndex*
  tilDataMeasGroupInfoTimeStampTime → *tilDataMeasGroupInfoTimeSectionStart*
- Improved: Unicode strings – It is now possible to use filenames with pure Unicode characters which do not belong to the active code page.
- Improved: *TilGetString*, *TilGetData* and *TilGetNextIteratorItems* are more fault-tolerant. *TilGetString* returns a truncated NUL-terminated string if the function fails due to an insufficient buffer. However, the respected string must be initialize (e.g. with NUL) before asking for a truncated string. *TilGetData* returns the truncated data if the data buffer is insufficient. *TilGetNextIteratorItems* returns the remaining samples successfully if the number of remaining samples is not null and lower than the requested number of samples.
- Fixed: *TilGetNextIteratorItems* for property *tilDataMeasGroupSignalDataIterator* behaved only as expected (beside a *false* return value for a non-zero iterator item size) if the requested number of samples matched the sample count. (A request for a lower number of samples yielded missing samples for succeeding calls of *TilGetNextIteratorItems*).

V1.1.1
- Fixed: *TilGetString* returned a wrong timestamp string in V1.1.0 (However, the function *TilGetTime64* works properly in all DLL versions.)

V1.1.0
- New: Property *tilDataMeasGroupInfoTimeSection* (then known as *tilDataMeasGroupInfoTimeStamp*)

V1.0.1
- New: Raw data iterator property (*tilDataMeasGroupSignalDataRawIterator*)
- New: *double* data iterator property (*tilDataMeasGroupSignalDataIterator*)

V1.0.0
- First release

	V1.0.0	V1.0.1	V1.1.0	V1.1.1	V1.2		V1.3	
					32-bit	x64	32-bit	x64
Minimum supported Windows version	Win95 or WinNT4	Win95 or WinNT4	Win95 or WinNT4	Win95 or WinNT4	Win95 or WinNT4	WinXP x64	Win95 or WinNT4	WinXP x64
Data type TIL_FLOAT80	X	X	X	X	X	-	X	-
Iterators	-	X	X	X	X	X	X	X
Multiple sections (= timestamps)	-	-	X	X	X	X	X	X
Pure Unicode characters	-	-	-	-	X	X	X	X
Fault-tolerant data block returns	-	-	-	-	X	X	X	X
x64 support	-	-	-	-	-	X	-	X
TMA support	-	-	-	-	-	-	X	X
***TilGetNextIteratorItems* for *double* data iterator with *count* < *sample count* works as expected**	-	-	-	-	X	X	X	X
***TilGetString* for *TILID_TIME64* works as expected**	X	X	-	X	X	X	X	X