



# Forschungsinstitut

umati Transformation Engine - API documentation

(Release Candidate, 2021-10-01)



<b>1 Introduction</b>	<b>1</b>
1.1 Recommended Reading . . . . .	1
<b>2 Initialization of a data client plugin</b>	<b>3</b>
<b>3 Known issues</b>	<b>5</b>
3.1 API definition issues . . . . .	5
3.2 Documentation/Style issues . . . . .	5
<b>4 Todo List</b>	<b>7</b>
<b>5 Module Index</b>	<b>9</b>
5.1 Modules . . . . .	9
<b>6 Data Structure Index</b>	<b>11</b>
6.1 Data Structures . . . . .	11
<b>7 File Index</b>	<b>13</b>
7.1 File List . . . . .	13
<b>8 Module Documentation</b>	<b>15</b>
8.1 Transformation Engine . . . . .	15
8.1.1 Detailed Description . . . . .	15
8.2 Data Client . . . . .	15
8.2.1 Detailed Description . . . . .	16
8.2.2 Data Structure Documentation . . . . .	16
8.2.2.1 struct tek_sa_data_client_capabilities . . . . .	16
8.2.3 Typedef Documentation . . . . .	17
8.2.3.1 tek_sa_data_client_handle . . . . .	17
8.2.3.2 tek_sa_load_plugin_fn . . . . .	17
8.2.4 Enumeration Type Documentation . . . . .	17
8.2.4.1 tek_sa_threading_model . . . . .	17
8.3 Common Definitions . . . . .	18
8.3.1 Detailed Description . . . . .	21
8.3.2 Data Structure Documentation . . . . .	21
8.3.2.1 struct tek_sa_additional_file . . . . .	21
8.3.2.2 struct tek_sa_configuration . . . . .	21
8.3.2.3 struct tek_sa_guid . . . . .	21
8.3.2.4 struct tek_sa_byte_string . . . . .	22
8.3.2.5 struct tek_sa_string . . . . .	22
8.3.2.6 struct tek_sa_complex_data . . . . .	22
8.3.2.7 struct tek_sa_complex_data_array_item . . . . .	23
8.3.2.8 struct tek_sa_complex_data_array . . . . .	23
8.3.2.9 struct tek_sa_complex_data_matrix . . . . .	23
8.3.2.10 struct tek_sa_variant_array . . . . .	24

8.3.2.11 struct tek_sa_variant_matrix . . . . .	24
8.3.2.12 struct tek_sa_variant . . . . .	25
8.3.2.13 struct tek_sa_struct_field_type_definition . . . . .	25
8.3.2.14 struct tek_sa_struct_definition . . . . .	25
8.3.2.15 struct tek_sa_enum_item_definition . . . . .	26
8.3.2.16 struct tek_sa_enum_definition . . . . .	26
8.3.2.17 struct tek_sa_method_argument_description . . . . .	26
8.3.2.18 struct tek_sa_field_write_request . . . . .	26
8.3.2.19 struct tek_sa_write_result . . . . .	27
8.3.2.20 struct tek_sa_read_result . . . . .	27
8.3.2.21 struct tek_sa_event_parameter . . . . .	27
8.3.2.22 struct tek_sa_dc_event . . . . .	27
8.3.2.23 union tek_sa_variant_array.data . . . . .	28
8.3.2.24 union tek_sa_variant.data . . . . .	29
8.3.3 Macro Definition Documentation . . . . .	29
8.3.3.1 TEK_SA_FIELD_HANDLE_INVALID . . . . .	29
8.3.3.2 TEK_SA_EVENT_HANDLE_INVALID . . . . .	30
8.3.3.3 TEK_SA_ALARM_HANDLE_INVALID . . . . .	30
8.3.3.4 TEK_SA_METHOD_HANDLE_INVALID . . . . .	30
8.3.3.5 TEK_SA_ERR_SUCCESS . . . . .	30
8.3.3.6 TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE . . . . .	31
8.3.3.7 TEK_SA_ERR_OUT_OF_MEMORY . . . . .	31
8.3.3.8 TEK_SA_ERR_INVALID_PARAMETER . . . . .	31
8.3.3.9 TEK_SA_ERR_RETRY_LATER . . . . .	31
8.3.3.10 TEK_SA_READ_RESULT_STATUS_OK . . . . .	32
8.3.3.11 TEK_SA_READ_RESULT_STATUS_NOK . . . . .	32
8.3.3.12 TEK_SA_READ_RESULT_STATUS_TIMEOUT . . . . .	32
8.3.3.13 TEK_SA_READ_RESULT_STATUS_INVALID_HANDLE . . . . .	32
8.3.3.14 TEK_SA_BLOCK_TRANSFER_END_OF_FILE . . . . .	32
8.3.3.15 TEK_SA_BLOCK_TRANSFER_ABORT . . . . .	33
8.3.4 Typedef Documentation . . . . .	33
8.3.4.1 tek_sa_type_handle . . . . .	33
8.3.4.2 tek_sa_type_handle_or_type_enum . . . . .	33
8.3.4.3 tek_sa_datetime . . . . .	33
8.3.4.4 tek_sa_field_value . . . . .	34
8.3.4.5 tek_sa_field_handle . . . . .	34
8.3.4.6 tek_sa_event_handle . . . . .	34
8.3.4.7 tek_sa_alarm_handle . . . . .	34
8.3.4.8 tek_sa_method_handle . . . . .	34
8.3.4.9 TEK_SA_RESULT . . . . .	35
8.3.5 Enumeration Type Documentation . . . . .	35
8.3.5.1 tek_sa_variant_type . . . . .	35

8.3.5.2 tek_sa_field_attributes . . . . .	36
8.3.5.3 tek_sa_log_level_t . . . . .	36
<b>9 Data Structure Documentation</b>	<b>37</b>
9.1 tek_sa_data_client Struct Reference . . . . .	37
9.1.1 Detailed Description . . . . .	38
9.1.2 Field Documentation . . . . .	38
9.1.2.1 register_features . . . . .	38
9.1.2.2 connect . . . . .	38
9.1.2.3 free . . . . .	39
9.1.2.4 read_fields . . . . .	39
9.1.2.5 write_fields . . . . .	41
9.1.2.6 block_read . . . . .	41
9.1.2.7 block_write . . . . .	42
9.1.2.8 subscribe . . . . .	42
9.1.2.9 unsubscribe . . . . .	43
9.1.2.10 invoke . . . . .	43
9.1.2.11 acknowledge_alarm . . . . .	44
9.1.2.12 handle . . . . .	44
9.2 tek_sa_data_client_plugin Struct Reference . . . . .	44
9.2.1 Detailed Description . . . . .	45
9.2.2 Field Documentation . . . . .	45
9.2.2.1 plugin_context . . . . .	45
9.2.2.2 data_client_new . . . . .	45
9.2.2.3 free_context . . . . .	46
9.3 tek_sa_transformation_engine Struct Reference . . . . .	46
9.3.1 Detailed Description . . . . .	47
9.3.2 Field Documentation . . . . .	48
9.3.2.1 register_field . . . . .	48
9.3.2.2 register_method . . . . .	48
9.3.2.3 register_event . . . . .	49
9.3.2.4 register_alarm . . . . .	49
9.3.2.5 register_enum_type . . . . .	50
9.3.2.6 register_struct_type . . . . .	50
9.3.2.7 post_event . . . . .	50
9.3.2.8 set_alarm . . . . .	51
9.3.2.9 reset_alarm . . . . .	51
9.3.2.10 log . . . . .	52
9.3.2.11 get_global_event . . . . .	52
9.3.2.12 update_capabilities . . . . .	53
9.3.2.13 read_progress . . . . .	53
9.3.2.14 read_result . . . . .	53

9.3.2.15 notify_change . . . . .	54
9.3.2.16 write_result . . . . .	54
9.3.2.17 call_method_result . . . . .	55
9.3.2.18 block_read_data . . . . .	55
9.3.2.19 block_write_data . . . . .	56
9.3.2.20 block_write_result . . . . .	56
<b>10 File Documentation</b>	<b>57</b>
10.1 include/south_api.h File Reference . . . . .	57
10.1.1 Detailed Description . . . . .	60
10.1.2 Macro Definition Documentation . . . . .	60
10.1.2.1 TEK_SA_API_VERSION_MAJOR . . . . .	61
10.1.2.2 TEK_SA_API_VERSION_MINOR . . . . .	61
10.1.2.3 TEK_SA_API_VERSION_PATCH . . . . .	61
10.1.2.4 TEK_SA_API_VERSION . . . . .	61
10.2 south_api.h . . . . .	61
<b>Index</b>	<b>69</b>

# Chapter 1

## Introduction

This documentation describes the interface between the umati Transformation Engine and its Data Clients.

### 1.1 Recommended Reading

- Start with [Initialization of a data client plugin](#) to get an overview of the relation between transformation engine, shared library, data\_client\_plugin and data\_client.
- Continue with the sections [Transformation Engine](#) and [Data Client](#) which contain the main components of the interface, namely [tek\\_sa\\_transformation\\_engine](#) and [tek\\_sa\\_data\\_client](#).





## Chapter 2

# Initialization of a data client plugin

Each data client shared library represents one plugin. One plugin may be responsible for multiple data client instances of (possibly) different type. Which type of data client is to be created is defined in the configuration. This configuration is passed to a call to [tek\\_sa\\_data\\_client\\_plugin::data\\_client\\_new](#).

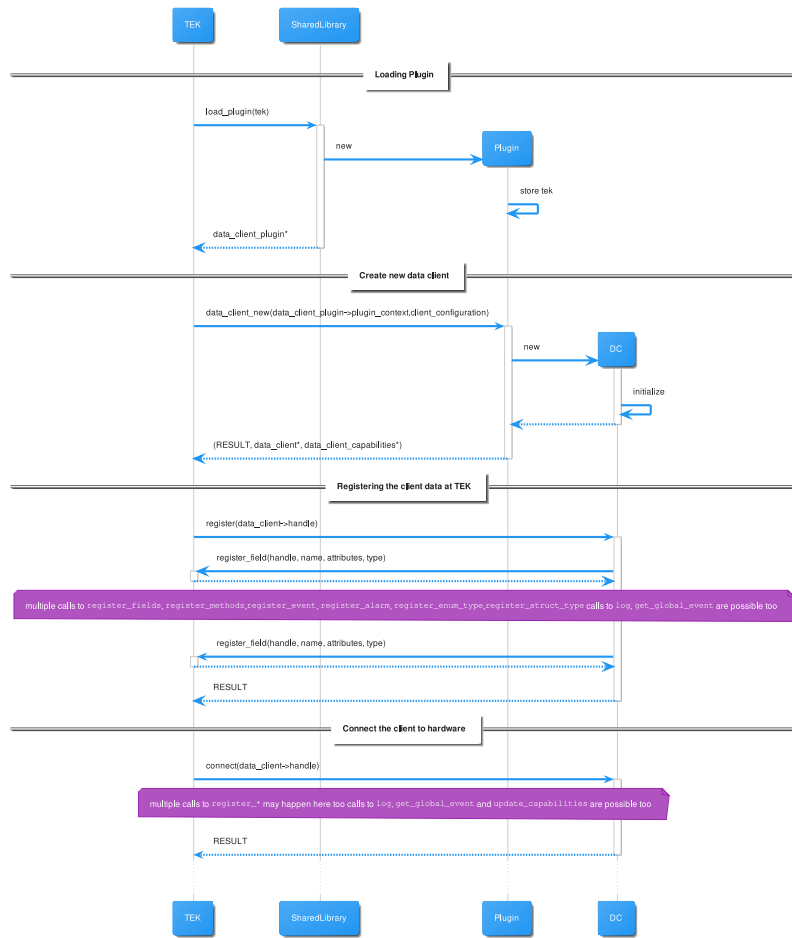
After loading the shared library the TEK calls the main initialization function with the fixed name `load_plugin` and a signature of [tek\\_sa\\_load\\_plugin\\_fn](#) . This function creates a new singleton instance of [tek\\_sa\\_data\\_client\\_plugin](#) and is expected to save the given TEK api struct.

Using the created [tek\\_sa\\_data\\_client\\_plugin](#), the TEK calls its [tek\\_sa\\_data\\_client\\_plugin::data\\_client\\_new](#) method for each configuration.

Each data client then is initialized with calls to [tek\\_sa\\_data\\_client::register\\_features](#) and [tek\\_sa\\_data\\_client::connect](#).

[tek\\_sa\\_data\\_client::register\\_features](#) should do all registration tasks which are possible without a connection to the hardware.

[tek\\_sa\\_data\\_client::connect](#) should connect to the hardware and register all new fields, types etc. Additionally it may happen that the capabilities of the data client change after connecting because more information about the hardware are known. Therefore it is expected that a call to [tek\\_sa\\_transformation\\_engine::update\\_capabilities](#) will happen.



## Chapter 3

# Known issues

### 3.1 API definition issues

This sections contains a list of yet unresolved issues concerning the definition of the API which do not relate directly to specific structs or functions.

**Todo** [B, JF] A struct `tek_configuration` is needed, which contains e.g. the global request timeout value.

**Todo** [D] A possibility to unregister fields, methods, events etc. is needed.

**Todo** [D] A possibility to define the sampling interval of subscribed fields is needed.

**Todo** [A, TEAM] What should be the datatype of the array dimension(s) (int32 or uint32)?

### 3.2 Documentation/Style issues

**Todo** [C, MIG] mkdocs/doxybook2 output can not handle union

**Todo** [C, MIG] mkdocs/doxybook2 output can not handle typedefs

**Todo** [C, MIG] mkdocs/doxybook2 output can not handle function pointers



# Chapter 4

## Todo List

### Page **Known issues**

- [D] A possibility to unregister fields, methods, events etc. is needed.
- [D] A possibility to define the sampling interval of subscribed fields is needed.
- [A, TEAM] What should be the datatype of the array dimension(s) (int32 or uint32)?
- [C, MIG] mkdocs/doxybook2 output can not handle union
- [C, MIG] mkdocs/doxybook2 output can not handle typedefs
- [C, MIG] mkdocs/doxybook2 output can not handle function pointers
- [B, JF] A struct `tek_configuration` is needed, which contains e.g. the global request timeout value.

### Class **tek\_sa\_complex\_data\_array**

- [B, TEAM] should this struct contain the number of bytes in `data` for sanity checks?

### Class **tek\_sa\_complex\_data\_matrix**

- [B, TEAM] should this struct contain the number of bytes in `data` for sanity checks?

Global **tek\_sa\_data\_client::read\_fields** )(tek\_sa\_data\_client\_handle dc, uint64\_t request\_id, const tek\_sa\_↵  
\_field\_handle items\_to\_read[], size\_t number\_of\_items, bool do\_not\_block)

- [B, TEAM] define error values of read function

Global **tek\_sa\_data\_client::subscribe** )(tek\_sa\_data\_client\_handle dc, const tek\_sa\_field\_handle items\_↵  
to\_subscribe[], size\_t number\_of\_items)

- [D, TEAM] add sampling rate parameter

Global **tek\_sa\_data\_client::write\_fields** )(tek\_sa\_data\_client\_handle dc, uint64\_t request\_id, const struct  
**tek\_sa\_field\_write\_request** items\_to\_write[], size\_t number\_of\_items, bool do\_not\_block)

- [B, TEAM] should the data client call a progress function if the operation needs more time?

### Class **tek\_sa\_transformation\_engine**

- [A, TEAM] inconsistent register\* methods signatures: always return error code or handle

Global **tek\_sa\_transformation\_engine::get\_global\_event** )(const char \*name)

- [C, TEAM] define the predefined events
- [C, TEAM] define return value when event with given name does not exist?

Global **tek\_sa\_transformation\_engine::read\_progress** )(tek\_sa\_data\_client\_handle dc, uint64\_t request\_id,  
uint64\_t progress)

- [B, TEAM] when should a data client report progress?
- [B, TEAM] when can the TEK stop the client (after progress was not reported)?

Global **tek\_sa\_transformation\_engine::set\_alarm** )(tek\_sa\_data\_client\_handle dc, const tek\_sa\_alarm\_↵  
handle alarm)

- [C, TEAM] called by data\_client after connect, regardless of "acknowledge" calls during previous connection?



## Chapter 5

# Module Index

### 5.1 Modules

Here is a list of all modules:

Transformation Engine . . . . .	15
Data Client . . . . .	15
Common Definitions . . . . .	18





## Chapter 6

# Data Structure Index

### 6.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">tek_sa_data_client</a>	
The interface of one instance of a data client . . . . .	37
<a href="#">tek_sa_data_client_plugin</a>	
Interface of the data client plugin . . . . .	44
<a href="#">tek_sa_transformation_engine</a>	
Interface of the Transformation Engine . . . . .	46



## Chapter 7

# File Index

### 7.1 File List

Here is a list of all files with brief descriptions:

include/[south\\_api.h](#)

Definition of the interface between Data Clients (DC) and the Transformation Engine (TEK) . . . [57](#)



## Chapter 8

# Module Documentation

### 8.1 Transformation Engine

#### Data Structures

- struct [tek\\_sa\\_transformation\\_engine](#)  
*Interface of the Transformation Engine.*

#### 8.1.1 Detailed Description

The module **Transformation Engine** contains the main API the transformation engine provides to data clients.

A client can interact the Transformation Engine API by accessing the *api* pointer which is given to the `load_plugin` function. (see the [tek\\_sa\\_load\\_plugin\\_fn](#) description)

Structs and definitions which are used in both the transformation engine and the data client API are described in the section [Common Definitions](#) .

### 8.2 Data Client

#### Data Structures

- struct [tek\\_sa\\_data\\_client\\_capabilities](#)  
*capabilities of the data client. These capabilities are applied to the complete data client as well as to each instance (device connection). [More...](#)*
- struct [tek\\_sa\\_data\\_client](#)  
*The interface of one instance of a data client.*
- struct [tek\\_sa\\_data\\_client\\_plugin](#)  
*Interface of the data client plugin.*

## Typedefs

- typedef void \* [tek\\_sa\\_data\\_client\\_handle](#)  
*The type of the data client handle.*
- typedef [TEK\\_SA\\_RESULT](#)(\* [tek\\_sa\\_load\\_plugin\\_fn](#)) (struct [tek\\_sa\\_transformation\\_engine](#) \*api, const struct [tek\\_sa\\_configuration](#) \*plugin\_configuration, struct [tek\\_sa\\_data\\_client\\_plugin](#) \*plugin)  
*Signature for the load plugin function.*

## Enumerations

- enum [tek\\_sa\\_threading\\_model](#) { [TEK\\_SA\\_THREADING\\_MODEL\\_SAME\\_THREAD](#) = 0x0 , [TEK\\_SA\\_THREADING\\_MODEL\\_SERIAL](#) = 0x1 , [TEK\\_SA\\_THREADING\\_MODEL\\_PARALLEL](#) = 0x2 }
- Describes the threading model of a data client instance of a data client plugin.*

### 8.2.1 Detailed Description

The module **Data Client** contains the API a data client has to implement. Optional parts of the interface are marked accordingly.

Structs and definitions which are used in both the transformation engine and the data client API are described in the section [Common Definitions](#) .

### 8.2.2 Data Structure Documentation

#### 8.2.2.1 struct [tek\\_sa\\_data\\_client\\_capabilities](#)

capabilities of the data client. These capabilities are applied to the complete data client as well as to each instance (device connection).

#### Remarks

As these capabilities are extended in the specification process it may be necessary to split the capabilities of the data client and the instance into different structs.

Definition at line [1008](#) of file [south\\_api.h](#).

#### Data Fields

<a href="#">size_t</a>	<a href="#">number_of_inflight_calls</a>	<p>Number of uncompleted async api calls. Unlimited number of uncompleted calls are signaled using 0 A blocking client uses 1 to signal that the TEK must wait for each result before requesting the next operation.</p> <p><b>Remarks</b></p> <p>This information may be dependent on the physical device and therefore available only after the connection was established.</p>
enum <a href="#">tek_sa_threading_model</a>	<a href="#">threading_model</a>	Requirements for the thread calling any communication function in the data client API.

## 8.2.3 Typedef Documentation

### 8.2.3.1 tek\_sa\_data\_client\_handle

```
typedef void* tek_sa_data_client_handle
```

The type of the data client handle.

An opaque handle for data client plugins. Internal structure of the data\_client implementation of a specific plugin is hidden behind this pointer.

Definition at line 215 of file [south\\_api.h](#).

### 8.2.3.2 tek\_sa\_load\_plugin\_fn

```
typedef TEK_SA_RESULT(* tek_sa_load_plugin_fn) (struct tek_sa_transformation_engine *api, const  
struct tek_sa_configuration *plugin_configuration, struct tek_sa_data_client_plugin *plugin)
```

Signature for the load plugin function.

The shared library of the data client will export the function 'load\_plugin' that fills a struct data\_client\_plugin.

#### Parameters

<i>api</i>	The TEK api.
<i>plugin_configuration</i>	Additional configuration files, e.g. licensing information, for the plugin itself.
<i>plugin</i>	The result of the initialized plugin.

#### Returns

Success or failure code.

Definition at line 1769 of file [south\\_api.h](#).

## 8.2.4 Enumeration Type Documentation

### 8.2.4.1 tek\_sa\_threading\_model

```
enum tek_sa_threading_model
```

Describes the threading model of a data client instance of a data client plugin.

## Enumerator

TEK_SA_THREADING_MODEL_SAME_THREAD	The same thread must always be used to call the data client instance.
TEK_SA_THREADING_MODEL_SEQUENTIAL	Only one thread of a thread pool is doing a single call at a time at the data client instance.
TEK_SA_THREADING_MODEL_PARALLEL	<p>DLL is thread safe, multiple parallel calls are allowed.</p> <p><b>Remarks</b></p> <p>If the number of parallel tasks in the data client is reached, the API call may return <code>ASYNC_RESULT_RETRY_LATER</code>.</p>

Definition at line 978 of file [south\\_api.h](#).

## 8.3 Common Definitions

### Data Structures

- struct [tek\\_sa\\_additional\\_file](#)  
Configuration class which describes an additional file which is passed to the data client. [More...](#)
- struct [tek\\_sa\\_configuration](#)  
Configuration object containing the contents of the configuration files for the [tek\\_sa\\_data\\_client\\_plugin](#) or [tek\\_sa\\_data\\_client](#) instances. [More...](#)
- struct [tek\\_sa\\_guid](#)  
The representation of a GUID when used as a field type. [More...](#)
- struct [tek\\_sa\\_byte\\_string](#)  
The representation of a byte array with variable length when used as a field type. [More...](#)
- struct [tek\\_sa\\_string](#)  
The representation of a string with variable length when used as a field type. [More...](#)
- struct [tek\\_sa\\_complex\\_data](#)  
The representation of a field value which has a type which is not a predefined type. [More...](#)
- struct [tek\\_sa\\_complex\\_data\\_array\\_item](#)  
The representation of the items of an array of complex data values with exactly one dimension. [More...](#)
- struct [tek\\_sa\\_complex\\_data\\_array](#)  
The representation of an array of complex data with exactly one dimension. [More...](#)
- struct [tek\\_sa\\_complex\\_data\\_matrix](#)  
The representation of array of complex data with more than one dimension. [More...](#)
- struct [tek\\_sa\\_variant\\_array](#)  
The representation of a one dimensional array of the supported base types. [More...](#)
- struct [tek\\_sa\\_variant\\_matrix](#)  
The representation of an array with more than one dimension of the supported base types. [More...](#)
- struct [tek\\_sa\\_variant](#)  
The representation of a single value (which may be of array type too). [More...](#)
- struct [tek\\_sa\\_struct\\_field\\_type\\_definition](#)  
The type definition of a record field in a user defined struct type. [More...](#)
- struct [tek\\_sa\\_struct\\_definition](#)  
The type definition of a user defined record type. [More...](#)
- struct [tek\\_sa\\_enum\\_item\\_definition](#)



- The definition of an enum item which is defined in a user defined enum type. [More...](#)

  - struct [tek\\_sa\\_enum\\_definition](#)

The type definition of a user defined enum type. [More...](#)
- struct [tek\\_sa\\_method\\_argument\\_description](#)

The description of a method parameter. [More...](#)
- struct [tek\\_sa\\_field\\_write\\_request](#)

Structure to encapsulate the parameters of a write field request. [More...](#)
- struct [tek\\_sa\\_write\\_result](#)

Structure to encapsulate the result of a write field request. [More...](#)
- struct [tek\\_sa\\_read\\_result](#)

Structure to encapsulate the result of a read operation of a single field. [More...](#)
- struct [tek\\_sa\\_event\\_parameter](#)

Structure to encapsulate an event parameter. [More...](#)
- struct [tek\\_sa\\_dc\\_event](#)

An event which may be sent from the data client to [tek\\_sa\\_transformation\\_engine::post\\_event](#). [More...](#)
- union [tek\\_sa\\_variant\\_array.data](#)

The array values. [More...](#)
- union [tek\\_sa\\_variant.data](#)

The value. [More...](#)

## Typedefs

- typedef int64\_t [tek\\_sa\\_type\\_handle](#)

The type of a handle which is returned for user defined types.
- typedef int64\_t [tek\\_sa\\_type\\_handle\\_or\\_type\\_enum](#)

The type for a reference handle which references either a user defined type (see [tek\\_sa\\_type\\_handle](#)) or a predefined type (See [tek\\_sa\\_variant\\_type](#).)
- typedef int64\_t [tek\\_sa\\_datetime](#)

The type of date and time values wen used as a field type.
- typedef struct [tek\\_sa\\_variant](#) [tek\\_sa\\_field\\_value](#)

Type of data client field values.
- typedef uint32\_t [tek\\_sa\\_field\\_handle](#)

Handle type for a field definition.
- typedef uint32\_t [tek\\_sa\\_event\\_handle](#)

Handle type for an event definition.
- typedef uint32\_t [tek\\_sa\\_alarm\\_handle](#)

Handle type for an alarm definition.
- typedef uint32\_t [tek\\_sa\\_method\\_handle](#)

Handle type for a method definition.

## Enumerations

- enum [tek\\_sa\\_variant\\_type](#) {  
[TEK\\_SA\\_VARIANT\\_TYPE\\_NULL](#) = 0x0 , [TEK\\_SA\\_VARIANT\\_TYPE\\_BOOL](#) = 0x1 , [TEK\\_SA\\_VARIANT\\_TYPE\\_UINT8\\_T](#) = 0x2 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT8\\_T](#) = 0x3 ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_UINT16\\_T](#) = 0x4 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT16\\_T](#) = 0x5 , [TEK\\_SA\\_VARIANT\\_TYPE\\_UINT32\\_T](#) = 0x6 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT32\\_T](#) = 0x7 ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_UINT64\\_T](#) = 0x8 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT64\\_T](#) = 0x9 , [TEK\\_SA\\_VARIANT\\_TYPE\\_FLOAT](#) = 0xa , [TEK\\_SA\\_VARIANT\\_TYPE\\_DOUBLE](#) = 0xb ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_DATETIME](#) = 0xc , [TEK\\_SA\\_VARIANT\\_TYPE\\_STRING](#) = 0xd , [TEK\\_SA\\_VARIANT\\_TYPE\\_GUID](#) = 0xe , [TEK\\_SA\\_VARIANT\\_TYPE\\_BYTE\\_STRING](#) = 0xf ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_COMPLEX](#) = 0x20 , [TEK\\_SA\\_VARIANT\\_TYPE\\_FLAG\\_ARRAY](#) = 0x40 ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_FLAG\\_MATRIX](#) = 0x80 }

The predefined types which can be processed in the TE.

- enum `tek_sa_field_attributes` { `TEK_SA_FIELD_ATTRIBUTES_WRITABLE` = 0x1 , `TEK_SA_FIELD_ATTRIBUTES_READABLE` = 0x2 , `TEK_SA_FIELD_ATTRIBUTES_SUBSCRIBABLE` = 0x4 }

Flags type which contains the attributes of a data client field.

- enum `tek_sa_log_level_t` { `TEK_SA_LOG_LEVEL_TRACE` = 0x0 , `TEK_SA_LOG_LEVEL_DEBUG` = 0x1 , `TEK_SA_LOG_LEVEL_INFO` = 0x2 , `TEK_SA_LOG_LEVEL_WARNING` = 0x3 , `TEK_SA_LOG_LEVEL_ERROR` = 0x4 , `TEK_SA_LOG_LEVEL_CRITICAL` = 0x5 }

Definition of the possible logging levels which can be used in `tek_sa_transformation_engine::log`.

## StatusCodes

- typedef int `TEK_SA_RESULT`

The return value type of all interface functions (which need to return information about success of the operation).

- #define `TEK_SA_ERR_SUCCESS` 0  
An operation was completed successfully.
- #define `TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE` 10  
A data client function was called in an asynchronous manner while the implementation can not use multiple threads.
- #define `TEK_SA_ERR_OUT_OF_MEMORY` 11  
The data client or the Transformation Engine can not process a request because it has no more system resources.
- #define `TEK_SA_ERR_INVALID_PARAMETER` 12  
The parameters passed to the function are invalid.
- #define `TEK_SA_ERR_RETRY_LATER` 0xffffffff  
A data client function was called in an asynchronous manner while the number of inflight calls is already active.
- #define `TEK_SA_READ_RESULT_STATUS_OK` 0  
A read operation completed successfully.
- #define `TEK_SA_READ_RESULT_STATUS_NOK` 1  
A read operation failed.
- #define `TEK_SA_READ_RESULT_STATUS_TIMEOUT` 2  
A read operation did not complete within the specified time limit.
- #define `TEK_SA_READ_RESULT_STATUS_INVALID_HANDLE` 3  
The read operation failed because the passed field handle was invalid.
- #define `TEK_SA_BLOCK_TRANSFER_END_OF_FILE` 26  
The read operation read until the end of file.
- #define `TEK_SA_BLOCK_TRANSFER_ABORT` 24  
The block read or write operation should be stopped.

## Handle Constants

- #define `TEK_SA_FIELD_HANDLE_INVALID` 0  
An always invalid field handle.
- #define `TEK_SA_EVENT_HANDLE_INVALID` 0  
An always invalid event handle.
- #define `TEK_SA_ALARM_HANDLE_INVALID` 0  
An always invalid alarm handle.
- #define `TEK_SA_METHOD_HANDLE_INVALID` 0  
An always invalid method handle.

### 8.3.1 Detailed Description

The module **Common Definitions** contains functions, structs and typedefs which are used by the [Data Client](#) as well as the [Transformation Engine](#).

### 8.3.2 Data Structure Documentation

#### 8.3.2.1 struct tek\_sa\_additional\_file

Configuration class which describes an additional file which is passed to the data client.

Definition at line 225 of file [south\\_api.h](#).

##### Data Fields

char *	name	The name of the additional file as written in the configuration.
char *	content	The content of additional file.

#### 8.3.2.2 struct tek\_sa\_configuration

Configuration object containing the contents of the configuration files for the [tek\\_sa\\_data\\_client\\_plugin](#) or [tek\\_sa\\_data\\_client](#) instances.

Definition at line 240 of file [south\\_api.h](#).

##### Data Fields

char *	config	The configuration file as UTF-8 encoded JSON string
struct <a href="#">tek_sa_additional_file</a> *	additional_files	The additional files which are referenced in the configuration.
size_t	additional_files_count	The number of additional files

#### 8.3.2.3 struct tek\_sa\_guid

The representation of a GUID when used as a field type.

built-in types (bool, (u)int\_{8,16,32,64}\_t, strings, guides, datetime; subset of <https://reference.opcfoundation.org/Core/docs/Part6/5.1.2/>

See also <https://reference.opcfoundation.org/v104/Core/docs/Part6/5.1.3/>

Definition at line 283 of file [south\\_api.h](#).

##### Data Fields

uint32_t	data1	The Data1 field.
uint16_t	data2	The Data2 field.
uint16_t	data3	The Data3 field.
uint8_t	data4[8]	The Data4 field.

### 8.3.2.4 struct tek\_sa\_byte\_string

The representation of a byte array with variable length when used as a field type.

See <https://reference.opcfoundation.org/Core/docs/Part6/5.2.2/#5.2.2.7>

Definition at line 308 of file [south\\_api.h](#).

#### Data Fields

int32_t	length	The length of the byte string.
unsigned char *	data	The bytes of the byte string

### 8.3.2.5 struct tek\_sa\_string

The representation of a string with variable length when used as a field type.

See <https://reference.opcfoundation.org/Core/docs/Part6/5.2.2/#5.2.2.4>

#### Attention

The string encoding is always UTF-8.

Definition at line 326 of file [south\\_api.h](#).

#### Data Fields

int32_t	length	The length of the byte string.
unsigned char *	data	The UTF-8 encoded characters of the string.

### 8.3.2.6 struct tek\_sa\_complex\_data

The representation of a field value which has a type which is not a predefined type.

A value with a complex data type which was registered at the tek by calling [tek\\_sa\\_transformation\\_engine::register\\_struct\\_type](#).

Definition at line 351 of file [south\\_api.h](#).

#### Data Fields

<a href="#">tek_sa_type_handle</a>	type	The type handle of the registered data type.
uint32_t	data_length	The number of bytes in the <code>data</code> field. This is needed because the encoded length may differ for items of the same type.
unsigned char *	data	The bytes of the serialized value. The serialization is compatible with the binary OPC UA encoding of structures as described in <a href="https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.6/">https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.6/</a> .

**8.3.2.7 struct tek\_sa\_complex\_data\_array\_item**

The representation of the items of an array of complex data values with exactly one dimension.

See also [tek\\_sa\\_complex\\_data\\_array](#)

Definition at line 381 of file [south\\_api.h](#).

**Data Fields**

uint32_t	data_length	The number of bytes in the <code>data</code> field. This is needed because the encoded length may differ for items of the same type.
unsigned char *	data	The bytes of the serialized value. See also <a href="#">tek_sa_complex_data::data</a>

**8.3.2.8 struct tek\_sa\_complex\_data\_array**

The representation of an array of complex data with exactly one dimension.

A one-dimensional array of values which are of a complex data type.

**Todo** [B, TEAM] should this struct contain the number of bytes in `data` for sanity checks?

Definition at line 410 of file [south\\_api.h](#).

**Data Fields**

<a href="#">tek_sa_type_handle</a>	type	The type handle of the registered type of the array items.
size_t	number_of_items	The number of items in the array.
struct <a href="#">tek_sa_complex_data_array_item</a> *	data	The array data, which consists of the concatenation of all serialized items.

**8.3.2.9 struct tek\_sa\_complex\_data\_matrix**

The representation of array of complex data with more than one dimension.

A multi-dimensional array of values which are of a complex data type.

**Todo** [B, TEAM] should this struct contain the number of bytes in `data` for sanity checks?

Definition at line 435 of file [south\\_api.h](#).

## Data Fields

<a href="#">tek_sa_type_handle</a>	type	The type handle of the registered type of the array items.
int32_t	dimension_length	The number of dimensions in the array.
int32_t *	dimensions	The array dimensions. Multi-dimensional arrays are encoded as a one-dimensional array and this field specifies the dimensions of the array. The original array can be reconstructed using this information. Higher rank dimensions are serialized first. For example, an array with dimensions [2,2,2] is written in this order: [0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1] This is compatible with the encoding used by OPC UA array types: <a href="https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.2/#5.2.2.16">https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.2/#5.2.2.16</a>
struct <a href="#">tek_sa_complex_data_array_item</a> *	data	The array data, which consists of the concatenation of all serialized items.

**8.3.2.10 struct tek\_sa\_variant\_array**

The representation of a one dimensional array of the supported base types.

Definition at line 538 of file [south\\_api.h](#).

## Data Fields

int32_t	length	The number of elements in the array.
union <a href="#">tek_sa_variant_array.data</a>	data	The array values.

**8.3.2.11 struct tek\_sa\_variant\_matrix**

The representation of an array with more than one dimension of the supported base types.

Definition at line 566 of file [south\\_api.h](#).

## Data Fields

int32_t	dimension_length	The number of array dimensions.
---------	------------------	---------------------------------

## Data Fields

int32_t *	dimensions	The array dimensions. Multi-dimensional arrays are encoded as a one-dimensional array and this field specifies the dimensions of the array. The original array can be reconstructed using this information. Higher rank dimensions are serialized first. For example, an array with dimensions [2,2,2] is written in this order: [0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1] This is compatible with the encoding used by OPC UA array types: <a href="https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.2/#5.2.2.16">https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.2/#5.2.2.16</a>
struct <a href="#">tek_sa_variant_array</a>	data	The array values.

**8.3.2.12 struct tek\_sa\_variant**

The representation of a single value (which may be of array type too).

Definition at line 593 of file [south\\_api.h](#).

## Data Fields

uint8_t	type	The type of the value. Must be one of the values described in <a href="#">tek_sa_variant_type</a> .
union <a href="#">tek_sa_variant.data</a>	data	The value.

**8.3.2.13 struct tek\_sa\_struct\_field\_type\_definition**

The type definition of a record field in a user defined struct type.

Definition at line 638 of file [south\\_api.h](#).

## Data Fields

char *	name	The name of the data field.
<a href="#">tek_sa_type_handle_or_type_enum</a>	type	The type of the field, represented as type_handle or type_enum.

**8.3.2.14 struct tek\_sa\_struct\_definition**

The type definition of a user defined record type.

Definition at line 651 of file [south\\_api.h](#).

## Data Fields

char *	name	The name of the type.
struct <a href="#">tek_sa_struct_field_type_definition</a> *	items	The definition of the record fields.
size_t	item_count	The number of fields in the record type.

### 8.3.2.15 struct tek\_sa\_enum\_item\_definition

The definition of an enum item which is defined in a user defined enum type.

Definition at line 669 of file [south\\_api.h](#).

#### Data Fields

char *	name	The name of the enum item.
int32_t	value	The numeric value of the enum item.

### 8.3.2.16 struct tek\_sa\_enum\_definition

The type definition of a user defined enum type.

Definition at line 682 of file [south\\_api.h](#).

#### Data Fields

char *	name	The name of the type.
struct <a href="#">tek_sa_enum_item_definition</a> *	items	The defined enum values of this type.
size_t	item_count	The number of defined enum values.

### 8.3.2.17 struct tek\_sa\_method\_argument\_description

The description of a method parameter.

See [tek\\_sa\\_transformation\\_engine::register\\_method](#)

Definition at line 701 of file [south\\_api.h](#).

#### Data Fields

char const *	name	The name of the method parameter.
enum <a href="#">tek_sa_variant_type</a>	type	The type of the method parameter.

### 8.3.2.18 struct tek\_sa\_field\_write\_request

Structure to encapsulate the parameters of a write field request.

Definition at line 841 of file [south\\_api.h](#).

#### Data Fields

<a href="#">tek_sa_field_handle</a>	handle	The field handle as returned from <a href="#">tek_sa_transformation_engine::register_field</a> .
<a href="#">tek_sa_field_value</a>	value	The value to be written to the field.



**8.3.2.19 struct tek\_sa\_write\_result**

Structure to encapsulate the result of a write field request.

Definition at line 853 of file [south\\_api.h](#).

**Data Fields**

<a href="#">TEK_SA_RESULT</a>	status	The write operation result.
<a href="#">tek_sa_field_handle</a>	handle	The handle of the field written.

**8.3.2.20 struct tek\_sa\_read\_result**

Structure to encapsulate the result of a read operation of a single field.

Definition at line 865 of file [south\\_api.h](#).

**Data Fields**

<a href="#">TEK_SA_RESULT</a>	status	The read operation result.
<a href="#">tek_sa_field_handle</a>	handle	The handle of the read field.
<a href="#">tek_sa_field_value</a>	value	The read value.  <b>Attention</b> Must not be accessed if the <code>status</code> is not <a href="#">TEK_SA_ERR_SUCCESS</a>

**8.3.2.21 struct tek\_sa\_event\_parameter**

Structure to encapsulate an event parameter.

Definition at line 885 of file [south\\_api.h](#).

**Data Fields**

<code>char const *</code>	name	The name of the parameter.
<a href="#">tek_sa_field_value</a>	value	The value of the event parameter.

**8.3.2.22 struct tek\_sa\_dc\_event**

An event which may be sent from the data client to [tek\\_sa\\_transformation\\_engine::post\\_event](#).

Definition at line 897 of file [south\\_api.h](#).

## Data Fields

<a href="#">tek_sa_datetime</a>	timestamp	<p>The Timestamp of the event.</p> <p><b>Remarks</b></p> <p>This should be the a value as close as possible to the actual occurrence of the event.</p>
int16_t	severity	<p>The severity level of the event.</p> <p>The severity is defined as in <a href="https://reference.opcfoundation.org/v104/Core/docs/Part5/6.4.2/">https://reference.opcfoundation.org/v104/Core/docs/Part5/6.4.2/</a> which is cited here:</p> <p>Severity is an indication of the urgency of the Event. This is also commonly called “priority”. Values will range from 1 to 1 000, with 1 being the lowest severity and 1 000 being the highest. Typically, a severity of 1 would indicate an Event which is informational in nature, while a value of 1 000 would indicate an Event of catastrophic nature, which could potentially result in severe financial loss or loss of life.</p>
<a href="#">tek_sa_event_handle</a>	event_type	<p>The event type handle as returned by the call to <a href="#">tek_sa_transformation_engine::register_event</a>.</p> <p><b>Attention</b></p> <p>This field must not be <a href="#">TEK_SA_EVENT_HANDLE_INVALID</a></p>
<a href="#">tek_sa_field_handle</a>	source	<p>The handle of the source of the event.</p> <p>The source of the event is a field in the data client. As not all events have a source, this field may be equal to <a href="#">TEK_SA_FIELD_HANDLE_INVALID</a>.</p>
size_t	number_of_parameters	The number of event parameters.
struct <a href="#">tek_sa_event_parameter</a> *	parameters	The event parameters.

## 8.3.2.23 union tek\_sa\_variant\_array.data

The array values.

Definition at line 543 of file [south\\_api.h](#).

## Data Fields

bool *	b	
uint8_t *	ui8	
int8_t *	i8	
uint16_t *	ui16	
int16_t *	i16	
uint32_t *	ui32	
int32_t *	i32	

## Data Fields

uint64_t *	ui64	
int64_t *	i64	
float *	f	
double *	d	
<a href="#">tek_sa_datetime</a> *	dt	
struct <a href="#">tek_sa_string</a> *	s	
struct <a href="#">tek_sa_guid</a> *	guid	
struct <a href="#">tek_sa_byte_string</a> *	bs	

8.3.2.24 union [tek\\_sa\\_variant.data](#)

The value.

Definition at line 602 of file [south\\_api.h](#).

## Data Fields

bool	b	
uint8_t	ui8	
int8_t	i8	
uint16_t	ui16	
int16_t	i16	
uint32_t	ui32	
int32_t	i32	
uint64_t	ui64	
int64_t	i64	
float	f	
double	d	
<a href="#">tek_sa_datetime</a>	dt	
struct <a href="#">tek_sa_string</a>	s	
struct <a href="#">tek_sa_guid</a>	guid	
struct <a href="#">tek_sa_byte_string</a>	bs	
struct <a href="#">tek_sa_variant_array</a>	array	
struct <a href="#">tek_sa_variant_matrix</a>	matrix	
struct <a href="#">tek_sa_complex_data</a>	complex	
struct <a href="#">tek_sa_complex_data_array</a>	complex_array	
struct <a href="#">tek_sa_complex_data_matrix</a>	complex_matrix	

## 8.3.3 Macro Definition Documentation

8.3.3.1 [TEK\\_SA\\_FIELD\\_HANDLE\\_INVALID](#)

```
#define TEK_SA_FIELD_HANDLE_INVALID 0
```

An always invalid field handle.

Definition at line 744 of file [south\\_api.h](#).

#### 8.3.3.2 TEK\_SA\_EVENT\_HANDLE\_INVALID

```
#define TEK_SA_EVENT_HANDLE_INVALID 0
```

An always invalid event handle.

Definition at line 747 of file [south\\_api.h](#).

#### 8.3.3.3 TEK\_SA\_ALARM\_HANDLE\_INVALID

```
#define TEK_SA_ALARM_HANDLE_INVALID 0
```

An always invalid alarm handle.

Definition at line 750 of file [south\\_api.h](#).

#### 8.3.3.4 TEK\_SA\_METHOD\_HANDLE\_INVALID

```
#define TEK_SA_METHOD_HANDLE_INVALID 0
```

An always invalid method handle.

Definition at line 753 of file [south\\_api.h](#).

#### 8.3.3.5 TEK\_SA\_ERR\_SUCCESS

```
#define TEK_SA_ERR_SUCCESS 0
```

An operation was completed successfully.

Definition at line 771 of file [south\\_api.h](#).

#### 8.3.3.6 TEK\_SA\_ERR\_NON\_BLOCKING\_IMPOSSIBLE

```
#define TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE 10
```

A data client function was called in an asynchronous manner while the implementation can not use multiple threads.

The TEK will call the function in a synchronous manner again.

See [Asynchronous Data Client calls](#) and [tek\\_sa\\_data\\_client\\_capabilities](#)

Definition at line 782 of file [south\\_api.h](#).

#### 8.3.3.7 TEK\_SA\_ERR\_OUT\_OF\_MEMORY

```
#define TEK_SA_ERR_OUT_OF_MEMORY 11
```

The data client or the Transformation Engine can not process a request because it has no more system resources.

Definition at line 788 of file [south\\_api.h](#).

#### 8.3.3.8 TEK\_SA\_ERR\_INVALID\_PARAMETER

```
#define TEK_SA_ERR_INVALID_PARAMETER 12
```

The parameters passed to the function are invalid.

Definition at line 791 of file [south\\_api.h](#).

#### 8.3.3.9 TEK\_SA\_ERR\_RETRY\_LATER

```
#define TEK_SA_ERR_RETRY_LATER 0xffffffff
```

A data client function was called in an asynchronous manner while the number of inflight calls is already active.

The TEK will call the function again at a later time.

See [Asynchronous Data Client calls](#) and [tek\\_sa\\_data\\_client\\_capabilities](#)

Definition at line 803 of file [south\\_api.h](#).

#### 8.3.3.10 TEK\_SA\_READ\_RESULT\_STATUS\_OK

```
#define TEK_SA_READ_RESULT_STATUS_OK 0
```

A read operation completed successfully.

Definition at line 806 of file [south\\_api.h](#).

#### 8.3.3.11 TEK\_SA\_READ\_RESULT\_STATUS\_NOK

```
#define TEK_SA_READ_RESULT_STATUS_NOK 1
```

A read operation failed.

Definition at line 809 of file [south\\_api.h](#).

#### 8.3.3.12 TEK\_SA\_READ\_RESULT\_STATUS\_TIMEOUT

```
#define TEK_SA_READ_RESULT_STATUS_TIMEOUT 2
```

A read operation did not complete within the specified time limit.

Definition at line 812 of file [south\\_api.h](#).

#### 8.3.3.13 TEK\_SA\_READ\_RESULT\_STATUS\_INVALID\_HANDLE

```
#define TEK_SA_READ_RESULT_STATUS_INVALID_HANDLE 3
```

The read operation failed because the passed field handle was invalid.

Definition at line 816 of file [south\\_api.h](#).

#### 8.3.3.14 TEK\_SA\_BLOCK\_TRANSFER\_END\_OF\_FILE

```
#define TEK_SA_BLOCK_TRANSFER_END_OF_FILE 26
```

The read operation read until the end of file.

This result value applies to the [tek\\_sa\\_transformation\\_engine::block\\_read\\_data](#) callback.

Definition at line 824 of file [south\\_api.h](#).

### 8.3.3.15 TEK\_SA\_BLOCK\_TRANSFER\_ABORT

```
#define TEK_SA_BLOCK_TRANSFER_ABORT 24
```

The block read or write operation should be stopped.

This result value applies to the [tek\\_sa\\_transformation\\_engine::block\\_read\\_data](#) and the [tek\\_sa\\_transformation\\_engine::block\\_write\\_data](#) callback.

Definition at line 833 of file [south\\_api.h](#).

## 8.3.4 Typedef Documentation

### 8.3.4.1 tek\_sa\_type\_handle

```
typedef int64_t tek_sa_type_handle
```

The type of a handle which is returned for user defined types.

The TEK creates a unique type handle for every type registered with a call to [tek\\_sa\\_transformation\\_engine::register\\_struct\\_type](#) or [tek\\_sa\\_transformation\\_engine::register\\_enum\\_type](#). The TEK also ensures that the value range of these handles does not overlap with [tek\\_sa\\_variant\\_type](#).

Definition at line 267 of file [south\\_api.h](#).

### 8.3.4.2 tek\_sa\_type\_handle\_or\_type\_enum

```
typedef int64_t tek_sa_type_handle_or_type_enum
```

The type for a reference handle which references either a user defined type (see [tek\\_sa\\_type\\_handle](#)) or a predefined type (See [tek\\_sa\\_variant\\_type](#).)

Definition at line 273 of file [south\\_api.h](#).

### 8.3.4.3 tek\_sa\_datetime

```
typedef int64_t tek_sa_datetime
```

The type of date and time values were used as a field type.

The definition is based on OPC UA DateTime (see <https://reference.opcfoundation.org/Core/docs/Part6/5.2.2/#5.2.2.5>)

Definition at line 342 of file [south\\_api.h](#).

#### 8.3.4.4 tek\_sa\_field\_value

```
typedef struct tek_sa_variant tek_sa_field_value
```

Type of data client field values.

Definition at line 629 of file [south\\_api.h](#).

#### 8.3.4.5 tek\_sa\_field\_handle

```
typedef uint32_t tek_sa_field_handle
```

Handle type for a field definition.

Definition at line 728 of file [south\\_api.h](#).

#### 8.3.4.6 tek\_sa\_event\_handle

```
typedef uint32_t tek_sa_event_handle
```

Handle type for an event definition.

Definition at line 731 of file [south\\_api.h](#).

#### 8.3.4.7 tek\_sa\_alarm\_handle

```
typedef uint32_t tek_sa_alarm_handle
```

Handle type for an alarm definition.

Definition at line 734 of file [south\\_api.h](#).

#### 8.3.4.8 tek\_sa\_method\_handle

```
typedef uint32_t tek_sa_method_handle
```

Handle type for a method definition.

Definition at line 737 of file [south\\_api.h](#).



### 8.3.4.9 TEK\_SA\_RESULT

```
typedef int TEK_SA_RESULT
```

The return value type of all interface functions (which need to return information about success of the operation).

Definition at line 768 of file [south\\_api.h](#).

## 8.3.5 Enumeration Type Documentation

### 8.3.5.1 tek\_sa\_variant\_type

```
enum tek_sa_variant_type
```

The predefined types which can be processed in the TE.

This enum type is a composition of enum and flag values. Each enum value (the ones *not* starting with "TEK\_SA\_VARIANT\_TYPE\_FLAG") may be combined with zero or one flags (the ones starting with "TEK\_SA\_VARIANT\_TYPE\_FLAG").

#### Enumerator

TEK_SA_VARIANT_TYPE_NULL	The invalid type id.
TEK_SA_VARIANT_TYPE_BOOL	The type id of a bool value.
TEK_SA_VARIANT_TYPE_UINT8_T	The type id of an unsigned byte value.
TEK_SA_VARIANT_TYPE_INT8_T	The type id of a signed byte value.
TEK_SA_VARIANT_TYPE_UINT16_T	The type id of an unsigned short value.
TEK_SA_VARIANT_TYPE_INT16_T	The type id of a signed short value.
TEK_SA_VARIANT_TYPE_UINT32_T	The type id of an unsigned 32bit integer value.
TEK_SA_VARIANT_TYPE_INT32_T	The type id of a signed 32bit integer value value.
TEK_SA_VARIANT_TYPE_UINT64_T	The type id of an unsigned 64bit integer value.
TEK_SA_VARIANT_TYPE_INT64_T	The type id of a signed 64bit integer value.
TEK_SA_VARIANT_TYPE_FLOAT	The type id of a 32bit floating point value.
TEK_SA_VARIANT_TYPE_DOUBLE	The type id of a 64bit floating point value.
TEK_SA_VARIANT_TYPE_DATETIME	The type id of a date and time value. See <a href="#">tek_sa_datetime</a> .
TEK_SA_VARIANT_TYPE_STRING	The type id of a string value. See <a href="#">tek_sa_string</a> .
TEK_SA_VARIANT_TYPE_GUID	The type id of a GUID value. See <a href="#">tek_sa_guid</a> .
TEK_SA_VARIANT_TYPE_BYTE_STRING	The type id of a byte string value. See <a href="#">tek_sa_byte_string</a> .
TEK_SA_VARIANT_TYPE_COMPLEX	The type id of a value with a complex data type. See <a href="#">tek_sa_transformation_engine::register_struct_type</a> .
TEK_SA_VARIANT_TYPE_FLAG_ARRAY	The flag which is set to declare an array with one dimension of the base type.
TEK_SA_VARIANT_TYPE_FLAG_MATRIX	The flag which is set to declare an array with more than one dimension of the base type.

Definition at line 474 of file [south\\_api.h](#).

### 8.3.5.2 tek\_sa\_field\_attributes

enum `tek_sa_field_attributes`

Flags type which contains the attributes of a data client field.

Enumerator

TEK_SA_FIELD_ATTRIBUTES_WRITABLE	The attribute to mark a field as writeable.
TEK_SA_FIELD_ATTRIBUTES_READABLE	The attribute to mark a field as readable.
TEK_SA_FIELD_ATTRIBUTES_SUBSCRIBABLE	The attribute to mark a field which can be subscribed to.

Definition at line 716 of file `south_api.h`.

### 8.3.5.3 tek\_sa\_log\_level\_t

enum `tek_sa_log_level_t`

Definition of the possible logging levels which can be used in `tek_sa_transformation_engine::log`.

Enumerator

TEK_SA_LOG_LEVEL_TRACE	
TEK_SA_LOG_LEVEL_DEBUG	
TEK_SA_LOG_LEVEL_INFO	
TEK_SA_LOG_LEVEL_WARNING	
TEK_SA_LOG_LEVEL_ERROR	
TEK_SA_LOG_LEVEL_CRITICAL	

Definition at line 955 of file `south_api.h`.

## Chapter 9

# Data Structure Documentation

### 9.1 tek\_sa\_data\_client Struct Reference

The interface of one instance of a data client.

```
#include <south_api.h>
```

#### Data Fields

##### Lifecycle functions

- [TEK\\_SA\\_RESULT](#)(\* [register\\_features](#) )(tek\_sa\_data\_client\_handle dc)  
*Register all known features of the data client.*
- [TEK\\_SA\\_RESULT](#)(\* [connect](#) )(tek\_sa\_data\_client\_handle dc)  
*Connect the data client to the data source.*
- void(\* [free](#) )(tek\_sa\_data\_client\_handle dc)  
*Frees the data client and releases all its resources.*

##### Data client functions

- [TEK\\_SA\\_RESULT](#)(\* [read\\_fields](#) )(tek\_sa\_data\_client\_handle dc, uint64\_t request\_id, const [tek\\_sa\\_field\\_handle](#) items\_to\_read[], size\_t number\_of\_items, bool do\_not\_block)  
*Function to read one or more fields from the data client. The call may be executed in a synchronous or asynchronous manner (See parameter [do\\_not\\_block](#)).*
- [TEK\\_SA\\_RESULT](#)(\* [write\\_fields](#) )(tek\_sa\_data\_client\_handle dc, uint64\_t request\_id, const struct [tek\\_sa\\_field\\_write\\_request](#) items\_to\_write[], size\_t number\_of\_items, bool do\_not\_block)  
*Function to write values to data client fields.*
- [TEK\\_SA\\_RESULT](#)(\* [block\\_read](#) )(const [tek\\_sa\\_data\\_client\\_handle](#) dc, uint64\_t request\_id, const char \*filepath, uint64\_t offset, int64\_t length, bool do\_not\_block, int64\_t \*filesize)  
*Starts a block transfer from the client to the TEK.*
- [TEK\\_SA\\_RESULT](#)(\* [block\\_write](#) )(const [tek\\_sa\\_data\\_client\\_handle](#) dc, uint64\_t request\_id, const char \*filepath, uint64\_t offset, int64\_t length, bool do\_not\_block)  
*Start a block transfer from the TEK to the data client.*
- [TEK\\_SA\\_RESULT](#)(\* [subscribe](#) )(tek\_sa\_data\_client\_handle dc, const [tek\\_sa\\_field\\_handle](#) items\_to\_subscribe[], size\_t number\_of\_items)  
*Subscribe to changes of one ore more data client fields.*
- [TEK\\_SA\\_RESULT](#)(\* [unsubscribe](#) )(tek\_sa\_data\_client\_handle dc, const [tek\\_sa\\_field\\_handle](#) items\_to\_unsubscribe[], size\_t number\_of\_items)  
*Unsubscribe to changes of one ore more data client fields.*

- `TEK_SA_RESULT(* invoke)(const tek_sa_data_client_handle dc, const tek_sa_method_handle method, uint64_t request_id, const tek_sa_field_value parameters[], const size_t number_of_parameters)`  
*Invoke a method on the data client.*
- `void(* acknowledge_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm)`  
*Acknowledge an alarm in the data client.*

### Data fields

- `tek_sa_data_client_handle handle`  
*The handle that is passed as first parameter in all functions of this interface.*

## 9.1.1 Detailed Description

The interface of one instance of a data client.

Definition at line 1034 of file `south_api.h`.

## 9.1.2 Field Documentation

### 9.1.2.1 register\_features

```
TEK_SA_RESULT(* tek_sa_data_client::register_features) (tek_sa_data_client_handle dc)
```

Register all known features of the data client.

#### Parameters

<i>dc</i>	data client handle features are registered for
-----------	--

This method is called from the TEK after the data client was created and before it will be connected. See also [Initialization of a data client plugin](#)

A data client implementation should evaluate the configuration (passed to `tek_sa_data_client_plugin::data_client_new`) and register all known types fields, events, methods and alarms.

A connection to the controller must not be established.

Definition at line 1052 of file `south_api.h`.

### 9.1.2.2 connect

```
TEK_SA_RESULT(* tek_sa_data_client::connect) (tek_sa_data_client_handle dc)
```

Connect the data client to the data source.

This method is called from the TEK after the data client has registered its features. See also [Initialization of a data client plugin](#).

A data client implementation should connect to the data source and register additional features and capabilities.

If the data client can not connect to the data source it should keep trying to connect after the method call completed but it should not block.

Definition at line 1066 of file [south\\_api.h](#).

### 9.1.2.3 free

```
void(* tek_sa_data_client::free) (tek_sa_data_client_handle dc)
```

Frees the data client and releases all its resources.

Should be called by the TEK.

Definition at line 1073 of file [south\\_api.h](#).

### 9.1.2.4 read\_fields

```
TEK_SA_RESULT(* tek_sa_data_client::read_fields) (tek_sa_data_client_handle dc, uint64_t request_id, const tek_sa_field_handle items_to_read[], size_t number_of_items, bool do_not_block)
```

Function to read one or more fields from the data client. The call may be executed in a synchronous or asynchronous manner (See parameter `do_not_block`).

The values of the requested fields are sent by calling the [tek\\_sa\\_transformation\\_engine::read\\_result](#) callback function. The data client must preserve the order of the fields in the results that are provided in [tek\\_sa\\_transformation\\_engine::read\\_result](#) callback.

If the time needed to retrieve the values is larger than half the global timeout value a data client must call the `vde_sa_tek_ap::read_progress` callback function.

#### Parameters

<i>dc</i>	The <a href="#">handle</a> of the data client as returned from <a href="#">tek_sa_data_client_plugin::data_client_new</a> .
<i>request_id</i>	A unique request identifier which is created by the TEK and must be passed to call to <a href="#">tek_sa_transformation_engine::read_result</a> and <a href="#">tek_sa_transformation_engine::read_progress</a> .
<i>items_to_read</i>	An array of field handles which describes the values the data client should read. See also function <a href="#">tek_sa_transformation_engine::register_field</a> .
<i>number_of_items</i>	The number of handles in the parameter <code>items_to_read</code> .
<i>do_not_block</i>	A boolean flag that, when set to <code>true</code> , tells the data client that it should return immediately and return the read field values later in another thread.

## Returns

**TEK\_SA\_ERR\_SUCCESS** when the call succeeded.

**TEK\_SA\_ERR\_NON\_BLOCKING\_IMPOSSIBLE** if `do_not_block` is set to `true` and the called data client is not able to do nonblocking calls. The TEK will retry with `do_not_block` set to `false`

**TEK\_SA\_ERR\_OUT\_OF\_MEMORY** when the data client can not allocate the data structures and resources to read the fields.

any other error which applies to the read function

**Todo** [B, TEAM] define error values of read function

## Attention

It is mandatory that the data client does not block when called with parameter `do_not_block` set to `true`.

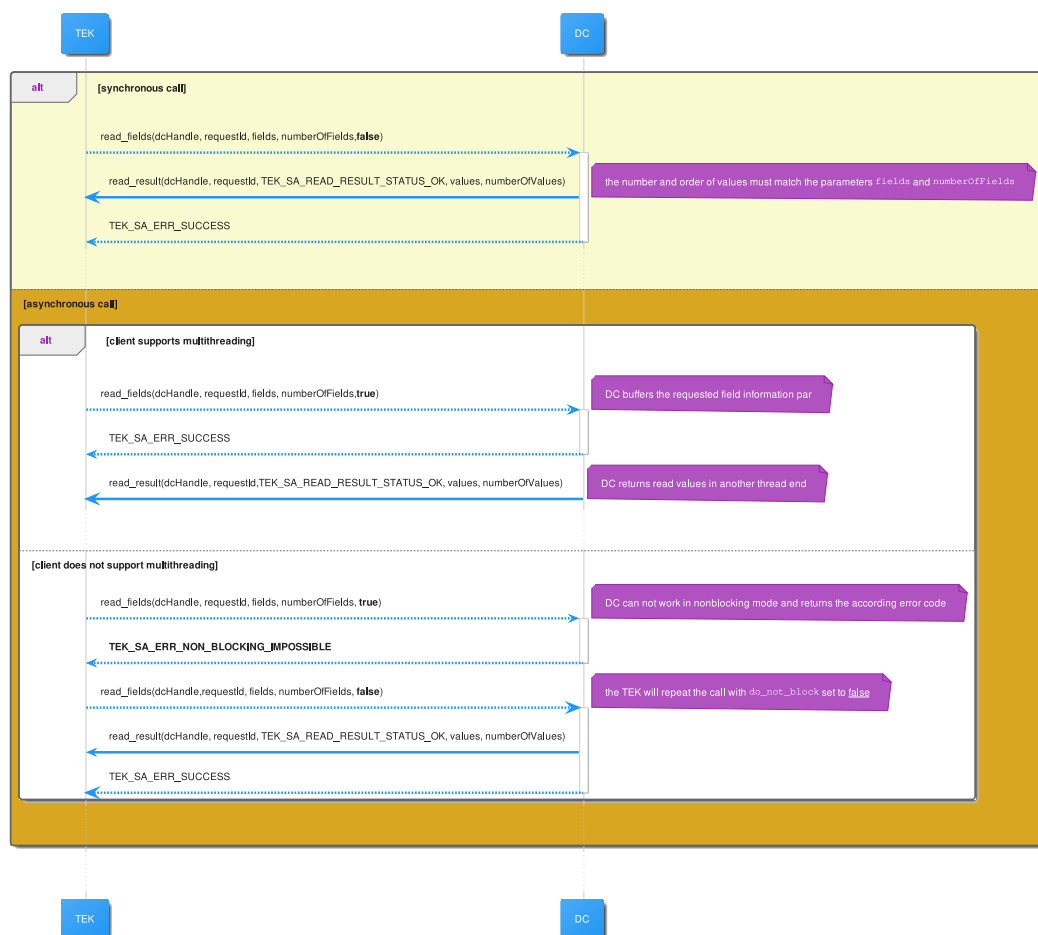
Usage of the Parameter `do_not_block`

Figure 9.1 Possible call sequences

Definition at line 1167 of file `south_api.h`.

## 9.1.2.5 write\_fields

```
TEK_SA_RESULT(* tek_sa_data_client::write_fields) (tek_sa_data_client_handle dc, uint64_t request_id, const struct tek_sa_field_write_request items_to_write[], size_t number_of_items, bool do_not_block)
```

Function to write values to data client fields.

## Parameters

<i>dc</i>	The <a href="#">handle</a> of the data client as returned from <a href="#">tek_sa_data_client_plugin::data_client_new</a> .
<i>request_id</i>	A unique request identifier which is created by the TEK and must be passed to call to <a href="#">tek_sa_transformation_engine::write_result</a> .
<i>items_to_write</i>	An array of field handles and their values which describes the values the data client should write.
<i>number_of_items</i>	The number of handles in the parameter <i>items_to_write</i> .
<i>do_not_block</i>	A boolean flag that, when set to <i>true</i> , tells the data client that it should return immediately and write the values in the background. See also <a href="#">Usage in read_fields</a>

**Todo** [B, TEAM] should the data client call a progress function if the operation needs more time?

Definition at line 1189 of file [south\\_api.h](#).

## 9.1.2.6 block\_read

```
TEK_SA_RESULT(* tek_sa_data_client::block_read) (const tek_sa_data_client_handle dc, uint64_t request_id, const char *filepath, uint64_t offset, int64_t length, bool do_not_block, int64_t *filesize)
```

Starts a block transfer from the client to the TEK.

For example, read a file from the device.

## Parameters

<i>dc</i>	The data client handle
<i>request_id</i>	The request id for the TEK API callbacks
<i>filepath</i>	The file or address of the block to be read. The format is data client specific. The pointer must be in utf-8.
<i>offset</i>	The offset in the data
<i>length</i>	A specific length, or -1 for the whole data
<i>do_not_block</i>	See <a href="#">Usage in read_fields</a>
<i>filesize</i>	The file size will be written by the data client, or -1 if not known at the call

**Returns**

An information about the success or failure of the operation.

The data is not yet passed to this method directly but sent from the data client in chunks to the [tek\\_sa\\_transformation\\_engine::block\\_read\\_data](#) callback.

Definition at line 1214 of file [south\\_api.h](#).

**9.1.2.7 block\_write**

```
TEK_SA_RESULT(* tek_sa_data_client::block_write) (const tek\_sa\_data\_client\_handle dc, uint64_t request_id, const char *filepath, uint64_t offset, int64_t length, bool do_not_block)
```

Start a block transfer from the TEK to the data client.

**Parameters**

<i>dc</i>	The <a href="#">handle</a> of the data client as returned from <a href="#">tek_sa_data_client_plugin::data_client_new</a> .
<i>request_id</i>	A unique request identifier which is created by the TEK and must be passed to call to <a href="#">tek_sa_transformation_engine::block_write_result</a> and <a href="#">tek_sa_transformation_engine::block_write_data</a> .
<i>offset</i>	The offset in the data
<i>length</i>	A specific length, or -1 for the whole data
<i>do_not_block</i>	See <a href="#">Usage in read_fields</a>

**Returns**

An information about the success or failure of the operation.

The data is not yet passed to this method directly but requested from the data client in chunks from the [tek\\_sa\\_transformation\\_engine::block\\_write\\_data](#) callback.

Definition at line 1237 of file [south\\_api.h](#).

**9.1.2.8 subscribe**

```
TEK_SA_RESULT(* tek_sa_data_client::subscribe) (tek\_sa\_data\_client\_handle dc, const tek\_sa\_field\_handle items_to_subscribe[], size_t number_of_items)
```

Subscribe to changes of one ore more data client fields.

**Parameters**

<i>dc</i>	The <a href="#">handle</a> of the data client as returned from <a href="#">tek_sa_data_client_plugin::data_client_new</a> .
<i>items_to_subscribe</i>	The fields for which change events will be received.
<i>number_of_items</i>	The number of elements in the <i>items_to_subscribe</i> parameter.



**Todo** [D, TEAM] add sampling rate parameter

The subscription mechanism is very easy compared to that of the OPC UA specification. The TEK can subscribe to each field only once and all changes are signaled by a call to the `tek_sa_data_transformation_engine::notify_↔` change callback.

Definition at line 1257 of file `south_api.h`.

### 9.1.2.9 unsubscribe

```
TEK_SA_RESULT(* tek_sa_data_client::unsubscribe) (tek_sa_data_client_handle dc, const tek_sa_field_handle
items_to_unsubscribe[], size_t number_of_items)
```

Unsubscribe to changes of one ore more data client fields.

#### Parameters

<i>dc</i>	The <a href="#">handle</a> of the data client as returned from <a href="#">tek_sa_data_client_plugin::data_client_new</a> .
<i>items_to_unsubscribe</i>	The fields for which no more change events will be received.
<i>number_of_items</i>	The number of elements in the <code>items_to_unsubscribe</code> parameter.

Definition at line 1270 of file `south_api.h`.

### 9.1.2.10 invoke

```
TEK_SA_RESULT(* tek_sa_data_client::invoke) (const tek_sa_data_client_handle dc, const tek_sa_method_handle
method, uint64_t request_id, const tek_sa_field_value parameters[], const size_t number_of_↔
parameters)
```

Invoke a method on the data client.

Providing this function ins optional

#### Parameters

<i>dc</i>	The <a href="#">handle</a> of the data client as returned from <a href="#">tek_sa_data_client_plugin::data_client_new</a> .
<i>method</i>	The method handle which is returned from the <code>tek_sa_data_transformation_engine::register_method</code> method.
<i>request_id</i>	A unique request identifier which is created by the TEK and must be passed to call to <a href="#">tek_sa_transformation_engine::block_write_result</a> and <a href="#">tek_sa_transformation_engine::block_write_data</a> .
<i>parameters</i>	The parameters of the method. Number and type must match the method registration.
<i>number_of_parameters</i>	The number of parameters in the <code>parameters</code> array.

The outcome of the message call is returned in the [tek\\_sa\\_transformation\\_engine::call\\_method\\_result](#) callback.

Definition at line 1296 of file [south\\_api.h](#).

#### 9.1.2.11 acknowledge\_alarm

```
void(* tek_sa_data_client::acknowledge_alarm) (tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm)
```

Acknowledge an alarm in the data client.

##### Parameters

<i>dc</i>	The <a href="#">handle</a> of the data client as returned from <a href="#">tek_sa_data_client_plugin::data_client_new</a> .
<i>alarm</i>	An alarm handle which is returned from the method <a href="#">tek_sa_transformation_engine::register_alarm</a> .

Called by TEK to signal triggered alarm has acknowledged by TEK consumer. The alarm may or may not be raised before with a call to [tek\\_sa\\_transformation\\_engine::set\\_alarm](#). When the alarm condition is not true anymore, then the data client implementation has to reset the alarm and call [tek\\_sa\\_transformation\\_engine::reset\\_alarm](#)

Definition at line 1315 of file [south\\_api.h](#).

#### 9.1.2.12 handle

```
tek_sa_data_client_handle tek_sa_data_client::handle
```

The handle that is passed as first parameter in all functions of this interface.

Definition at line 1327 of file [south\\_api.h](#).

The documentation for this struct was generated from the following file:

- [include/south\\_api.h](#)

## 9.2 tek\_sa\_data\_client\_plugin Struct Reference

Interface of the data client plugin.

```
#include <south_api.h>
```

## Data Fields

- void \* [plugin\\_context](#)  
*The (private) plugin context. Must be freed using free\_context on unloading the plugin.*
- [TEK\\_SA\\_RESULT](#)(\* [data\\_client\\_new](#))(void \*[plugin\\_context](#), const struct [tek\\_sa\\_configuration](#) \*config, struct [tek\\_sa\\_data\\_client](#) \*created\_client, struct [tek\\_sa\\_data\\_client\\_capabilities](#) \*capabilities)  
*Allocates and initializes the data client with a configuration. Prepare callbacks in data\_client.*
- void(\* [free\\_context](#))(void \*[plugin\\_context](#))  
*Frees the private context of the plugin.*

### 9.2.1 Detailed Description

Interface of the data client plugin.

The data client plugin is created once as result of a call to the `load_plugin` method();

Definition at line 1351 of file [south\\_api.h](#).

### 9.2.2 Field Documentation

#### 9.2.2.1 plugin\_context

```
void* tek_sa_data_client_plugin::plugin_context
```

The (private) plugin context. Must be freed using `free_context` on unloading the plugin.

Definition at line 1356 of file [south\\_api.h](#).

#### 9.2.2.2 data\_client\_new

```
TEK\_SA\_RESULT(* tek\_sa\_data\_client\_plugin::data\_client\_new) (void *plugin\_context, const struct tek\_sa\_configuration *config, struct tek\_sa\_data\_client *created_client, struct tek\_sa\_data\_client\_capabilities *capabilities)
```

Allocates and initializes the data client with a configuration. Prepare callbacks in `data_client`.

Does not perform any actions like connecting to the data source or register information at the TEK.

#### Parameters

<i>plugin_context</i>	
<i>config</i>	
<i>created_client</i>	
<i>capabilities</i>	The data client capabilities (known before connect), e.g. the threading model of the data client. Capabilities can be updated by the client using the TEK API, if additional information are retrieved later in the lifecycle of the data client.

**Returns**

failure code or success

Definition at line 1374 of file [south\\_api.h](#).

**9.2.2.3 free\_context**

```
void(* tek_sa_data_client_plugin::free_context) (void *plugin_context)
```

Frees the private context of the plugin.

Definition at line 1381 of file [south\\_api.h](#).

The documentation for this struct was generated from the following file:

- [include/south\\_api.h](#)

**9.3 tek\_sa\_transformation\_engine Struct Reference**

Interface of the Transformation Engine.

```
#include <south_api.h>
```

**Data Fields****Registration functions for data client operations and data fields**

- [tek\\_sa\\_field\\_handle](#)(\* [register\\_field](#) )(tek\_sa\_data\_client\_handle dc, const char \*name, enum [tek\\_sa\\_field\\_attributes](#) attributes, enum [tek\\_sa\\_variant\\_type](#) type)  
*Registers a new field of a data client with a name inside the TEK.*
- [tek\\_sa\\_method\\_handle](#)(\* [register\\_method](#) )(tek\_sa\_data\_client\_handle dc, const char \*name, struct [tek\\_sa\\_method\\_argument\\_description](#) input\_parameter[], size\_t number\_of\_input\_parameters, struct [tek\\_sa\\_method\\_argument\\_description](#) output\_parameter[], size\_t number\_of\_output\_parameters)  
*Registers a new method at the TEK.*
- [tek\\_sa\\_event\\_handle](#)(\* [register\\_event](#) )(tek\_sa\_data\_client\_handle dc, const char \*name)  
*Registers a new Event that a data client might raise.*
- [tek\\_sa\\_alarm\\_handle](#)(\* [register\\_alarm](#) )(tek\_sa\_data\_client\_handle dc, const char \*name, const int16\_t severity, const [tek\\_sa\\_field\\_handle](#) source)  
*Registers an alarm at the TEK.*

**Registration functions for extended types**

- [TEK\\_SA\\_RESULT](#)(\* [register\\_enum\\_type](#) )(tek\_sa\_data\_client\_handle dc, struct [tek\\_sa\\_enum\\_definition](#) const \*type\_definition, [tek\\_sa\\_type\\_handle](#) \*result)  
*Register a user defined enum type.*
- [TEK\\_SA\\_RESULT](#)(\* [register\\_struct\\_type](#) )(tek\_sa\_data\_client\_handle dc, struct [tek\\_sa\\_struct\\_definition](#) const \*type\_definition, [tek\\_sa\\_type\\_handle](#) \*result)  
*Register a user defined struct type.*

### Alarm and Event functions

- `TEK_SA_RESULT>(* post_event)(tek_sa_data_client_handle dc, struct tek_sa_dc_event const *event)`  
*Post an event which was declared with a call to either `get_global_event` or `register_event`.*
- `TEK_SA_RESULT(* set_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm)`  
*Sets an alarm.*
- `TEK_SA_RESULT(* reset_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm)`  
*Clears/resets an alarm.*

### Miscellaneous functions

- `void(* log)(tek_sa_data_client_handle source, enum tek_sa_log_level_t lvl, const char *format, va_list args)`  
*Logging function for data clients.*
- `tek_sa_event_handle(* get_global_event)(const char *name)`  
*Get a handle of a globally defined event.*
- `void(* update_capabilities)(tek_sa_data_client_handle dc, struct tek_sa_data_client_capabilities const *capabilities)`  
*Notifies the TEK of the change of the client's capabilities.*

### Data client callbacks

- `void(* read_progress)(tek_sa_data_client_handle dc, uint64_t request_id, uint64_t progress)`  
*Callback to signal progress of a read operation to the TEK.*
- `void(* read_result)(tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT result, const struct tek_sa_read_result results[], size_t number_of_results)`  
*Callback of the data client read operation.*
- `void(* notify_change)(tek_sa_data_client_handle dc, const struct tek_sa_read_result changes[], size_t number_of_changes)`  
*Callback to notify about a change of subscribed data fields.*
- `void(* write_result)(tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT result, const struct tek_sa_write_result results[], size_t number_of_results)`  
*Callback of the data client write operation.*
- `void(* call_method_result)(tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT result, const tek_sa_field_value results[], size_t number_of_results)`  
*Callback of a data client method call.*
- `TEK_SA_RESULT(* block_read_data)(tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT result, unsigned char buffer[], size_t buffer_length)`  
*Callback from the data client to the TEK signaling the next data chunk of the block transfer.*
- `TEK_SA_RESULT(* block_write_data)(tek_sa_data_client_handle dc, uint64_t request_id, unsigned char buffer[], size_t buffer_length, size_t *bytes_written)`  
*Callback from the data client to the TEK requesting another chunk to write to the data client.*
- `void(* block_write_result)(tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT result)`  
*Callback from the data client to the TEK with the final result of the block transfer.*

## 9.3.1 Detailed Description

Interface of the Transformation Engine.

Interface exported by the TEK, which is given a data client plugin (dll/so) to interact with the TEK.

**Todo** [A, TEAM] inconsistent register\* methods signatures: always return error code or handle

Definition at line 1399 of file `south_api.h`.

## 9.3.2 Field Documentation

### 9.3.2.1 register\_field

```
tek_sa_field_handle(* tek_sa_transformation_engine::register_field) (tek_sa_data_client_handle
dc, const char *name, enum tek_sa_field_attributes attributes, enum tek_sa_variant_type type)
```

Registers a new field of a data client with a name inside the TEK.

#### Parameters

<i>dc</i>	The data client that registers at the TEK.
<i>name</i>	The name of the field. The data client decides the name.
<i>attributes</i>	The attributes of the field, e.g. is writeable.
<i>type</i>	The data type of the field.

#### Returns

A valid field\_handle or INVALID\_FIELD\_HANDLE.

Definition at line 1413 of file [south\\_api.h](#).

### 9.3.2.2 register\_method

```
tek_sa_method_handle(* tek_sa_transformation_engine::register_method) (tek_sa_data_client_handle
dc, const char *name, struct tek_sa_method_argument_description input_parameter[], size_t
number_of_input_parameters, struct tek_sa_method_argument_description output_parameter[],
size_t number_of_output_parameters)
```

Registers a new method at the TEK.

#### Parameters

<i>dc</i>	The data client that registers at the TEK.
<i>name</i>	The name of the method.
<a href="#">tek_sa_method_argument_description</a>	The description of the method input arguments.
<i>number_of_input_parameters</i>	The number of input parameters.
<a href="#">tek_sa_method_argument_description</a>	The description of the method output arguments.
<i>number_of_output_parameters</i>	The number of output parameters.

#### Returns

A tek\_sa\_method\_handle.

Definition at line 1431 of file [south\\_api.h](#).

### 9.3.2.3 register\_event

```
tek_sa_event_handle(* tek_sa_transformation_engine::register_event) (tek_sa_data_client_handle  
dc, const char *name)
```

Registers a new Event that a data client might raise.

#### Parameters

<i>dc</i>	The data client that registers at the TEK.
<i>name</i>	The name of the event. Must be unique within all events registered from this dc.

#### Returns

A `tek_sa_event_handle` or `TEK_SA_EVENT_HANDLE_INVALID`, if the event registration failed (e.g. duplicate registration, empty name...).

The TEK ensures that the set of handles between the predefined events and the registered events are disjoint.

Definition at line 1450 of file [south\\_api.h](#).

### 9.3.2.4 register\_alarm

```
tek_sa_alarm_handle(* tek_sa_transformation_engine::register_alarm) (tek_sa_data_client_handle  
dc, const char *name, const intl6_t severity, const tek_sa_field_handle source)
```

Registers an alarm at the TEK.

#### Parameters

<i>dc</i>	The data client that registers at the TEK.
<i>name</i>	The name of the new alarm, must be unique within all alarms registered for this data client.
<i>severity</i>	The alarm severity level.
<i>source</i>	field the alarm relates to, the same field can be used for multiple alarms.

#### Returns

A `tek_sa_alarm_handle` or `TEK_SA_ALARM_HANDLE_INVALID`, if the alarm registration failed (e.g. duplicate registration, empty name...).

Definition at line 1466 of file [south\\_api.h](#).

### 9.3.2.5 register\_enum\_type

```
TEK_SA_RESULT(* tek_sa_transformation_engine::register_enum_type) (tek_sa_data_client_handle  
dc, struct tek_sa_enum_definition const *type_definition, tek_sa_type_handle *result)
```

Register a user defined enum type.

#### Parameters

<i>dc</i>	The data client that registers at the TEK.
<a href="#">tek_sa_enum_definition</a>	The definition of the enumeration.
<i>result</i>	A tek_sa_type_handle associated to the registered enum.

#### Returns

indicator whether the type definition was successfully registered

Definition at line 1486 of file [south\\_api.h](#).

### 9.3.2.6 register\_struct\_type

```
TEK_SA_RESULT(* tek_sa_transformation_engine::register_struct_type) (tek_sa_data_client_handle  
dc, struct tek_sa_struct_definition const *type_definition, tek_sa_type_handle *result)
```

Register a user defined struct type.

#### Parameters

<i>dc</i>	The data client that registers at the TEK.
<a href="#">tek_sa_struct_definition</a>	The definition of the struct.
<i>result</i>	A tek_sa_type_handle associated to the registered struct.

#### Returns

indicator whether the type definition was successfully registered

Definition at line 1499 of file [south\\_api.h](#).

### 9.3.2.7 post\_event

```
TEK_SA_RESULT(* tek_sa_transformation_engine::post_event) (tek_sa_data_client_handle dc, struct  
tek_sa_dc_event const *event)
```

Post an event which was declared with a call to either [get\\_global\\_event](#) or [register\\_event](#).



## Parameters

<i>dc</i>	Handle of the data client which sends the event.
<i>event</i>	A event structure. See <code>dc_event</code> .

## Returns

indicator whether the event was successfully posted or not

Definition at line 1519 of file [south\\_api.h](#).

### 9.3.2.8 set\_alarm

```
TEK_SA_RESULT(* tek_sa_transformation_engine::set_alarm) (tek_sa_data_client_handle dc, const  
tek_sa_alarm_handle alarm)
```

Sets an alarm.

## Parameters

<i>dc</i>	Handle of the data client that sets the alarm.
<i>alarm</i>	Handle of the alarm to be set.

## Returns

indicator whether setting the alarm was successful or not

**Todo** [C, TEAM] called by data\_client after connect, regardless of "acknowledge" calls during previous connection?

Definition at line 1532 of file [south\\_api.h](#).

### 9.3.2.9 reset\_alarm

```
TEK_SA_RESULT(* tek_sa_transformation_engine::reset_alarm) (tek_sa_data_client_handle dc,  
const tek_sa_alarm_handle alarm)
```

Clears/resets an alarm.

## Parameters

<i>dc</i>	Handle of the data client that clears/resets the alarm.
<i>alarm</i>	Handle of the alarm to be cleared/reset.

**Returns**

indicator whether resetting the alarm was successful or not

Definition at line 1542 of file [south\\_api.h](#).

**9.3.2.10 log**

```
void(* tek_sa_transformation_engine::log) (tek_sa_data_client_handle source, enum tek_sa_log_level_t
lvl, const char *format, va_list args)
```

Logging function for data clients.

The TEK bundles the messages of all data clients.

The TEK must be aware of data clients running in different threads than the TEK itself and is responsible for handling multi-threaded access to the function.

**Parameters**

<i>data_client_handle</i>	The data client that logs a message.
<i>lvl</i>	The logging level.
<i>format</i>	The message format string. Format must be compatible to <code>printf</code> .
<i>args</i>	A <code>va_list</code> that contains all the arguments for the format string.

Definition at line 1566 of file [south\\_api.h](#).

**9.3.2.11 get\_global\_event**

```
tek_sa_event_handle(* tek_sa_transformation_engine::get_global_event) (const char *name)
```

Get a handle of a globally defined event.

**Parameters**

<i>name</i>	name of globally defined event.
-------------	---------------------------------

**Returns**

handle to globally defined event

**Todo** [C, TEAM] define the predefined events

[C, TEAM] define return value when event with given name does not exist?

The TEK ensures that the set of handles between the predefined events and the registered events are disjoint.

Definition at line 1582 of file [south\\_api.h](#).

### 9.3.2.12 update\_capabilities

```
void(* tek_sa_transformation_engine::update_capabilities) (tek_sa_data_client_handle dc, struct
tek_sa_data_client_capabilities const *capabilities)
```

Notifies the TEK of the change of the client's capabilities.

#### Parameters

<i>dc</i>	Handle of the data client that informs about the change of its capabilities.
<i>tek_sa_data_client_capabilities</i>	The updated client capabilities.

#### Returns

(void)

Definition at line 1591 of file [south\\_api.h](#).

### 9.3.2.13 read\_progress

```
void(* tek_sa_transformation_engine::read_progress) (tek_sa_data_client_handle dc, uint64_t
request_id, uint64_t progress)
```

Callback to signal progress of a read operation to the TEK.

#### Parameters

<i>dc</i>	Handle of the data client that is the source of the call
<i>request_id</i>	id of request to data client which triggered the call back
<i>progress</i>	?? (percentage? why uint64?)

**Todo** [B, TEAM] when should a data client report progress?

**Todo** [B, TEAM] when can the TEK stop the client (after progress was not reported)?

Definition at line 1614 of file [south\\_api.h](#).

### 9.3.2.14 read\_result

```
void(* tek_sa_transformation_engine::read_result) (tek_sa_data_client_handle dc, uint64_t
request_id, TEK_SA_RESULT result, const struct tek_sa_read_result results[], size_t number_of_results)
```

Callback of the data client read operation.

**Parameters**

<i>dc</i>	Handle of the data client that is the source of the call
<i>request_id</i>	id of request to data client that triggered the call back
<i>result</i>	status code for read request
<i>results</i>	read values
<i>number_of_results</i>	length of results array

If the result is success, then the following constraints must hold:

The number of results MUST be equal to the number of fields requested in `read_fields`. The order of results MUST be the same as the order of fields in `read_fields`. The results array is only valid during the execution of the callback.

If the result is failure, the TEK MUST ignore the results and `number_of_results` parameters.

Definition at line 1638 of file `south_api.h`.

**9.3.2.15 notify\_change**

```
void(* tek_sa_transformation_engine::notify_change) (tek_sa_data_client_handle dc, const struct
tek_sa_read_result changes[], size_t number_of_changes)
```

Callback to notify about a change of subscribed data fields.

**Parameters**

<i>dc</i>	Handle of the data client that is the source of the change
<i>changes</i>	changed field values
<i>number_of_changes</i>	length of changes array

Definition at line 1650 of file `south_api.h`.

**9.3.2.16 write\_result**

```
void(* tek_sa_transformation_engine::write_result) (tek_sa_data_client_handle dc, uint64_t
request_id, TEK_SA_RESULT result, const struct tek_sa_write_result results[], size_t number↔
_of_results)
```

Callback of the data client write operation.

**Parameters**

<i>dc</i>	Handle of the data client data was written to
<i>request_id</i>	id of write request to data client that triggered the call back
<i>result</i>	overall result of write operation
<i>results</i>	write results for each written field
<i>number_of_results</i>	length of results array

Definition at line 1664 of file [south\\_api.h](#).

### 9.3.2.17 call\_method\_result

```
void(* tek_sa_transformation_engine::call_method_result) (tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT result, const tek_sa_field_value results[], size_t number_of_results)
```

Callback of a data client method call.

#### Parameters

<i>dc</i>	Handle of the data client a method was called at
<i>request_id</i>	id of method call request to data client that triggered the call back
<i>result</i>	error/success indicator of method call
<i>results</i>	return values of method call, only valid for successful results
<i>number_of_results</i>	length of results array

Definition at line 1680 of file [south\\_api.h](#).

### 9.3.2.18 block\_read\_data

```
TEK_SA_RESULT(* tek_sa_transformation_engine::block_read_data) (tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT result, unsigned char buffer[], size_t buffer_length)
```

Callback from the data client to the TEK signaling the next data chunk of the block transfer.

#### Parameters

<i>dc</i>	The data client handle.
<i>request_id</i>	The request id of the block transfer.
<i>result</i>	The data client signals success, error, or end-of-file. Buffer may contain a last chunk when end-of-file is signalled. If an error is signalled, the data client has aborted the process and will not call this callback again for the request.
<i>buffer</i>	The current chunk of the file. The TEK must copy the data into it's own process.
<i>buffer_length</i>	The length of the chunk.

#### Returns

The TEK responds with success, or can abort the transfer.

Definition at line 1701 of file [south\\_api.h](#).

### 9.3.2.19 block\_write\_data

```
TEK_SA_RESULT(* tek_sa_transformation_engine::block_write_data) (tek_sa_data_client_handle dc,
uint64_t request_id, unsigned char buffer[], size_t buffer_length, size_t *bytes_written)
```

Callback from the data client to the TEK requesting another chunk to write to the data client.

#### Parameters

<i>dc</i>	The data client handle.
<i>request_id</i>	The request id of the block transfer.
<i>buffer</i>	The buffer to write the chunk of the file. The TEK must copy the data into the buffer provided by the data client.
<i>buffer_length</i>	The length of the buffer in the data client.
<i>bytes_written</i>	The number of bytes written in the buffer by the TEK.
<i>result</i>	Signals valid next chunk, end-of-file, abort or error.

#### Returns

Success or failure code.

Definition at line 1718 of file [south\\_api.h](#).

### 9.3.2.20 block\_write\_result

```
void(* tek_sa_transformation_engine::block_write_result) (tek_sa_data_client_handle dc, uint64↔_t request_id, TEK_SA_RESULT result)
```

Callback from the data client to the TEK with the final result of the block transfer.

#### Parameters

<i>dc</i>	The data client handle.
<i>request↔_id</i>	The request id of the block transfer.
<i>result</i>	The final result.

Definition at line 1729 of file [south\\_api.h](#).

The documentation for this struct was generated from the following file:

- [include/south\\_api.h](#)

# Chapter 10

## File Documentation

### 10.1 include/south\_api.h File Reference

Definition of the interface between Data Clients (DC) and the Transformation Engine (TEK)

```
#include <stdarg.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
```

#### Data Structures

- struct [tek\\_sa\\_additional\\_file](#)  
*Configuration class which describes an additional file which is passed to the data client. [More...](#)*
- struct [tek\\_sa\\_configuration](#)  
*Configuration object containing the contents of the configuration files for the [tek\\_sa\\_data\\_client\\_plugin](#) or [tek\\_sa\\_data\\_client](#) instances. [More...](#)*
- struct [tek\\_sa\\_guid](#)  
*The representation of a GUID when used as a field type. [More...](#)*
- struct [tek\\_sa\\_byte\\_string](#)  
*The representation of a byte array with variable length when used as a field type. [More...](#)*
- struct [tek\\_sa\\_string](#)  
*The representation of a string with variable length when used as a field type. [More...](#)*
- struct [tek\\_sa\\_complex\\_data](#)  
*The representation of a field value which has a type which is not a predefined type. [More...](#)*
- struct [tek\\_sa\\_complex\\_data\\_array\\_item](#)  
*The representation of the items of an array of complex data values with exactly one dimension. [More...](#)*
- struct [tek\\_sa\\_complex\\_data\\_array](#)  
*The representation of an array of complex data with exactly one dimension. [More...](#)*
- struct [tek\\_sa\\_complex\\_data\\_matrix](#)  
*The representation of array of complex data with more than one dimension. [More...](#)*
- struct [tek\\_sa\\_variant\\_array](#)  
*The representation of a one dimensional array of the supported base types. [More...](#)*
- struct [tek\\_sa\\_variant\\_matrix](#)

- The representation of an array with more than one dimension of the supported base types. [More...](#)

  - struct [tek\\_sa\\_variant](#)
- The representation of a single value (which may be of array type too). [More...](#)

  - struct [tek\\_sa\\_struct\\_field\\_type\\_definition](#)
- The type definition of a record field in a user defined struct type. [More...](#)

  - struct [tek\\_sa\\_struct\\_definition](#)
- The type definition of a user defined record type. [More...](#)

  - struct [tek\\_sa\\_enum\\_item\\_definition](#)
- The definition of an enum item which is defined in a user defined enum type. [More...](#)

  - struct [tek\\_sa\\_enum\\_definition](#)
- The type definition of a user defined enum type. [More...](#)

  - struct [tek\\_sa\\_method\\_argument\\_description](#)
- The description of a method parameter. [More...](#)

  - struct [tek\\_sa\\_field\\_write\\_request](#)
- Structure to encapsulate the parameters of a write field request. [More...](#)

  - struct [tek\\_sa\\_write\\_result](#)
- Structure to encapsulate the result of a write field request. [More...](#)

  - struct [tek\\_sa\\_read\\_result](#)
- Structure to encapsulate the result of a read operation of a single field. [More...](#)

  - struct [tek\\_sa\\_event\\_parameter](#)
- Structure to encapsulate an event parameter. [More...](#)

  - struct [tek\\_sa\\_dc\\_event](#)
- An event which may be sent from the data client to [tek\\_sa\\_transformation\\_engine::post\\_event](#). [More...](#)

  - struct [tek\\_sa\\_data\\_client\\_capabilities](#)
- capabilities of the data client. These capabilities are applied to the complete data client as well as to each instance (device connection). [More...](#)

  - struct [tek\\_sa\\_data\\_client](#)
- The interface of one instance of a data client.

  - struct [tek\\_sa\\_data\\_client\\_plugin](#)
- Interface of the data client plugin.

  - struct [tek\\_sa\\_transformation\\_engine](#)
- Interface of the Transformation Engine.

  - union [tek\\_sa\\_variant\\_array.data](#)
- The array values. [More...](#)

  - union [tek\\_sa\\_variant.data](#)
- The value. [More...](#)

## Macros

- #define [TEK\\_SA\\_API\\_VERSION\\_MAJOR](#) 0
- #define [TEK\\_SA\\_API\\_VERSION\\_MINOR](#) 1
- #define [TEK\\_SA\\_API\\_VERSION\\_PATCH](#) 0
- #define [TEK\\_SA\\_API\\_VERSION](#) "0.1.0"

## Handle Constants

- #define [TEK\\_SA\\_FIELD\\_HANDLE\\_INVALID](#) 0
- An always invalid field handle.
- #define [TEK\\_SA\\_EVENT\\_HANDLE\\_INVALID](#) 0
- An always invalid event handle.
- #define [TEK\\_SA\\_ALARM\\_HANDLE\\_INVALID](#) 0
- An always invalid alarm handle.
- #define [TEK\\_SA\\_METHOD\\_HANDLE\\_INVALID](#) 0
- An always invalid method handle.



## Typedefs

- typedef void \* [tek\\_sa\\_data\\_client\\_handle](#)  
*The type of the data client handle.*
- typedef int64\_t [tek\\_sa\\_type\\_handle](#)  
*The type of a handle which is returned for user defined types.*
- typedef int64\_t [tek\\_sa\\_type\\_handle\\_or\\_type\\_enum](#)  
*The type for a reference handle which references either a user defined type (see [tek\\_sa\\_type\\_handle](#)) or a predefined type (See [tek\\_sa\\_variant\\_type](#).)*
- typedef int64\_t [tek\\_sa\\_datetime](#)  
*The type of date and time values wen used as a field type.*
- typedef struct [tek\\_sa\\_variant](#) [tek\\_sa\\_field\\_value](#)  
*Type of data client field values.*
- typedef uint32\_t [tek\\_sa\\_field\\_handle](#)  
*Handle type for a field definition.*
- typedef uint32\_t [tek\\_sa\\_event\\_handle](#)  
*Handle type for an event definition.*
- typedef uint32\_t [tek\\_sa\\_alarm\\_handle](#)  
*Handle type for an alarm definition.*
- typedef uint32\_t [tek\\_sa\\_method\\_handle](#)  
*Handle type for a method definition.*
- typedef [TEK\\_SA\\_RESULT](#)(\* [tek\\_sa\\_load\\_plugin\\_fn](#)) (struct [tek\\_sa\\_transformation\\_engine](#) \*api, const struct [tek\\_sa\\_configuration](#) \*plugin\_configuration, struct [tek\\_sa\\_data\\_client\\_plugin](#) \*plugin)  
*Signature for the load plugin function.*

## Enumerations

- enum [tek\\_sa\\_variant\\_type](#) {  
[TEK\\_SA\\_VARIANT\\_TYPE\\_NULL](#) = 0x0 , [TEK\\_SA\\_VARIANT\\_TYPE\\_BOOL](#) = 0x1 , [TEK\\_SA\\_VARIANT\\_TYPE\\_UINT8\\_T](#) = 0x2 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT8\\_T](#) = 0x3 ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_UINT16\\_T](#) = 0x4 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT16\\_T](#) = 0x5 , [TEK\\_SA\\_VARIANT\\_TYPE\\_UINT32\\_T](#) = 0x6 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT32\\_T](#) = 0x7 ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_UINT64\\_T](#) = 0x8 , [TEK\\_SA\\_VARIANT\\_TYPE\\_INT64\\_T](#) = 0x9 , [TEK\\_SA\\_VARIANT\\_TYPE\\_FLOAT](#) = 0xa , [TEK\\_SA\\_VARIANT\\_TYPE\\_DOUBLE](#) = 0xb ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_DATETIME](#) = 0xc , [TEK\\_SA\\_VARIANT\\_TYPE\\_STRING](#) = 0xd , [TEK\\_SA\\_VARIANT\\_TYPE\\_GUID](#) = 0xe , [TEK\\_SA\\_VARIANT\\_TYPE\\_BYTE\\_STRING](#) = 0xf ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_COMPLEX](#) = 0x20 , [TEK\\_SA\\_VARIANT\\_TYPE\\_FLAG\\_ARRAY](#) = 0x40 ,  
[TEK\\_SA\\_VARIANT\\_TYPE\\_FLAG\\_MATRIX](#) = 0x80 }  
*The predefined types which can be processed in the TE.*
- enum [tek\\_sa\\_field\\_attributes](#) { [TEK\\_SA\\_FIELD\\_ATTRIBUTES\\_WRITABLE](#) = 0x1 , [TEK\\_SA\\_FIELD\\_ATTRIBUTES\\_READABLE](#) = 0x2 , [TEK\\_SA\\_FIELD\\_ATTRIBUTES\\_SUBSCRIBABLE](#) = 0x4 }  
*Flags type which contains the attributes of a data client field.*
- enum [tek\\_sa\\_log\\_level\\_t](#) {  
[TEK\\_SA\\_LOG\\_LEVEL\\_TRACE](#) = 0x0 , [TEK\\_SA\\_LOG\\_LEVEL\\_DEBUG](#) = 0x1 , [TEK\\_SA\\_LOG\\_LEVEL\\_INFO](#) = 0x2 , [TEK\\_SA\\_LOG\\_LEVEL\\_WARNING](#) = 0x3 ,  
[TEK\\_SA\\_LOG\\_LEVEL\\_ERROR](#) = 0x4 , [TEK\\_SA\\_LOG\\_LEVEL\\_CRITICAL](#) = 0x5 }  
*Definition of the possible logging levels which can be used in [tek\\_sa\\_transformation\\_engine::log](#).*
- enum [tek\\_sa\\_threading\\_model](#) { [TEK\\_SA\\_THREADING\\_MODEL\\_SAME\\_THREAD](#) = 0x0 , [TEK\\_SA\\_THREADING\\_MODEL\\_S](#) = 0x1 , [TEK\\_SA\\_THREADING\\_MODEL\\_PARALLEL](#) = 0x2 }  
*Describes the threading model of a data client instance of a data client plugin.*

## StatusCodes

- `#define TEK_SA_ERR_SUCCESS 0`  
*An operation was completed successfully.*
- `#define TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE 10`  
*A data client function was called in an asynchronous manner while the implementation can not use multiple threads.*
- `#define TEK_SA_ERR_OUT_OF_MEMORY 11`  
*The data client or the Transformation Engine can not process a request because it has no more system resources.*
- `#define TEK_SA_ERR_INVALID_PARAMETER 12`  
*The parameters passed to the function are invalid.*
- `#define TEK_SA_ERR_RETRY_LATER 0xffffffff`  
*A data client function was called in an asynchronous manner while the number of inflight calls is already active.*
- `#define TEK_SA_READ_RESULT_STATUS_OK 0`  
*A read operation completed successfully.*
- `#define TEK_SA_READ_RESULT_STATUS_NOK 1`  
*A read operation failed.*
- `#define TEK_SA_READ_RESULT_STATUS_TIMEOUT 2`  
*A read operation did not complete within the specified time limit.*
- `#define TEK_SA_READ_RESULT_STATUS_INVALID_HANDLE 3`  
*The read operation failed because the passed field handle was invalid.*
- `#define TEK_SA_BLOCK_TRANSFER_END_OF_FILE 26`  
*The read operation read until the end of file.*
- `#define TEK_SA_BLOCK_TRANSFER_ABORT 24`  
*The block read or write operation should be stopped.*
- `typedef int TEK_SA_RESULT`  
*The return value type of all interface functions (which need to return information about success of the operation).*

### 10.1.1 Detailed Description

Definition of the interface between Data Clients (DC) and the Transformation Engine (TEK)

This header file conforms to the following standards:

- ISO/IEC 9899:1990 (C90)
- ISO/IEC 14882:1998 (C++98)

To ensure binary compatibility of the interface between different compilers and different versions of the interface, the struct offset of each struct member is verified at compile time. This check is realized by the `TEK_SA_VERIFY↵_STRUCT_OFFSET` macro.

Definition in file [south\\_api.h](#).

### 10.1.2 Macro Definition Documentation

**10.1.2.1 TEK\_SA\_API\_VERSION\_MAJOR**

```
#define TEK_SA_API_VERSION_MAJOR 0
```

Definition at line 19 of file [south\\_api.h](#).

**10.1.2.2 TEK\_SA\_API\_VERSION\_MINOR**

```
#define TEK_SA_API_VERSION_MINOR 1
```

Definition at line 20 of file [south\\_api.h](#).

**10.1.2.3 TEK\_SA\_API\_VERSION\_PATCH**

```
#define TEK_SA_API_VERSION_PATCH 0
```

Definition at line 21 of file [south\\_api.h](#).

**10.1.2.4 TEK\_SA\_API\_VERSION**

```
#define TEK_SA_API_VERSION "0.1.0"
```

Definition at line 22 of file [south\\_api.h](#).

**10.2 south\_api.h**

[Go to the documentation of this file.](#)

```
00001 #ifndef TEK_SOUTH_API_H
00002 #define TEK_SOUTH_API_H
00003
00019 #define TEK_SA_API_VERSION_MAJOR 0
00020 #define TEK_SA_API_VERSION_MINOR 1
00021 #define TEK_SA_API_VERSION_PATCH 0
00022 #define TEK_SA_API_VERSION "0.1.0"
00023
00024 #include <stdarg.h>
00025 #include <stddef.h>
00026 #include <stdbool.h>
00027 #include <stdint.h>
00028 #include <stdlib.h>
00029
00030
00031 #define TEK_SA_STRUCT_ALIGN_SELECT(O32, O64) (sizeof(void*) == 8 ? O64 : O32)
00032
00033 #if defined __STDC_VERSION__ && __STDC_VERSION__ >= 201112L
00034 #include <assert.h>
00035 #define TEK_SA_VERIFY_STRUCT_OFFSET(S, M, O32, O64) ; \
00036     _Static_assert(offsetof(struct S, M) == TEK_SA_STRUCT_ALIGN_SELECT(O32, O64), "struct offset of \
    field \"#M\" in \"#S\" must be correct")
00037 #else
00038 #define TEK_SA_VERIFY_STRUCT_OFFSET(S, M, O32, O64) ; \
00039     enum { S##__##M##_offset = 1/(int) (!! (offsetof(struct S, M) == TEK_SA_STRUCT_ALIGN_SELECT(O32, \
    O64))) };

```

```

00040 #endif
00041
00042 #ifdef __cplusplus
00043 extern "C" {
00044 #endif
00045
00215 typedef void* tek_sa_data_client_handle;
00216
00217 /*****
00218  * Configuration structures
00219  *****/
00220
00225 struct tek_sa_additional_file {
00227     char* name;
00228
00230     char* content;
00231 };
00232
00233 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_additional_file, name, 0, 0);
00234 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_additional_file, content, 4, 8);
00235
00240 struct tek_sa_configuration {
00242     char* config;
00243
00245     struct tek_sa_additional_file* additional_files;
00246
00248     size_t additional_files_count;
00249 };
00250
00251 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, config, 0, 0);
00252 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, additional_files, 4, 8);
00253 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, additional_files_count, 8, 16);
00254
00255 /*****
00256  * Built-in type definitions and variant
00257  *****/
00258
00267 typedef int64_t tek_sa_type_handle;
00268
00273 typedef int64_t tek_sa_type_handle_or_type_enum;
00274
00283 struct tek_sa_guid {
00285     uint32_t data1;
00286
00288     uint16_t data2;
00289
00291     uint16_t data3;
00292
00294     uint8_t data4[8];
00295 };
00296 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data1, 0, 0);
00297 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data2, 4, 4);
00298 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data3, 6, 6);
00299 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data4, 8, 8);
00300
00308 struct tek_sa_byte_string {
00310     int32_t length;
00311
00313     unsigned char* data;
00314 };
00315 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_byte_string, length, 0, 0);
00316 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_byte_string, data, 4, 8);
00317
00326 struct tek_sa_string {
00328     int32_t length;
00329
00331     unsigned char* data;
00332 };
00333 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_string, length, 0, 0);
00334 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_string, data, 4, 8);
00335
00342 typedef int64_t tek_sa_datetime;
00343
00351 struct tek_sa_complex_data {
00353     tek_sa_type_handle type;
00354
00361     uint32_t data_length;
00362
00369     unsigned char* data;
00370 };
00371 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, type, 0, 0);
00372 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, data_length, 8, 8);
00373 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, data, 12, 16);
00374
00381 struct tek_sa_complex_data_array_item {
00389     uint32_t data_length;
00390

```

```

00396 unsigned char* data;
00397 };
00398 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array_item, data_length, 0, 0);
00399 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array_item, data, 4, 8);
00400
00410 struct tek_sa_complex_data_array {
00412     tek_sa_type_handle type;
00413
00415     size_t number_of_items;
00416
00419     struct tek_sa_complex_data_array_item* data;
00420 };
00421
00422 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, type, 0, 0);
00423 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, number_of_items, 8, 8);
00424 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, data, 12, 16);
00425
00435 struct tek_sa_complex_data_matrix {
00437     tek_sa_type_handle type;
00438
00440     int32_t dimension_length;
00441
00456     int32_t* dimensions;
00457
00460     struct tek_sa_complex_data_array_item* data;
00461 };
00462 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, type, 0, 0);
00463 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, dimension_length, 8, 8);
00464 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, dimensions, 12, 16);
00465 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, data, 16, 24);
00466
00474 enum tek_sa_variant_type {
00476     TEK_SA_VARIANT_TYPE_NULL = 0x0,
00477
00479     TEK_SA_VARIANT_TYPE_BOOL = 0x1,
00480
00482     TEK_SA_VARIANT_TYPE_UINT8_T = 0x2,
00483
00485     TEK_SA_VARIANT_TYPE_INT8_T = 0x3,
00486
00488     TEK_SA_VARIANT_TYPE_UINT16_T = 0x4,
00489
00491     TEK_SA_VARIANT_TYPE_INT16_T = 0x5,
00492
00494     TEK_SA_VARIANT_TYPE_UINT32_T = 0x6,
00495
00497     TEK_SA_VARIANT_TYPE_INT32_T = 0x7,
00498
00500     TEK_SA_VARIANT_TYPE_UINT64_T = 0x8,
00501
00503     TEK_SA_VARIANT_TYPE_INT64_T = 0x9,
00504
00506     TEK_SA_VARIANT_TYPE_FLOAT = 0xa,
00507
00509     TEK_SA_VARIANT_TYPE_DOUBLE = 0xb,
00510
00512     TEK_SA_VARIANT_TYPE_DATETIME = 0xc,
00513
00515     TEK_SA_VARIANT_TYPE_STRING = 0xd,
00516
00518     TEK_SA_VARIANT_TYPE_GUID = 0xe,
00519
00521     TEK_SA_VARIANT_TYPE_BYTE_STRING = 0xf,
00522
00525     TEK_SA_VARIANT_TYPE_COMPLEX = 0x20,
00526
00529     TEK_SA_VARIANT_TYPE_FLAG_ARRAY = 0x40,
00530
00533     TEK_SA_VARIANT_TYPE_FLAG_MATRIX = 0x80
00534 };
00535
00538 struct tek_sa_variant_array {
00540     int32_t length;
00541
00543     union {
00544         bool* b;
00545         uint8_t* ui8;
00546         int8_t* i8;
00547         uint16_t* ui16;
00548         int16_t* i16;
00549         uint32_t* ui32;
00550         int32_t* i32;
00551         uint64_t* ui64;
00552         int64_t* i64;
00553         float* f;
00554         double* d;
00555         tek_sa_datetime* dt;

```

```

00556     struct tek_sa_string* s;
00557     struct tek_sa_guid* guid;
00558     struct tek_sa_byte_string* bs;
00559 } data;
00560 };
00561 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_array, length, 0, 0);
00562 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_array, data, 4, 8);
00563
00566 struct tek_sa_variant_matrix {
00568     int32_t dimension_length;
00569
00583     int32_t* dimensions;
00584
00586     struct tek_sa_variant_array data;
00587 };
00588 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_matrix, dimension_length, 0, 0);
00589 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_matrix, dimensions, 4, 8);
00590
00593 struct tek_sa_variant {
00599     uint8_t type;
00600
00602     union {
00603         bool b;
00604         uint8_t ui8;
00605         int8_t i8;
00606         uint16_t ui16;
00607         int16_t i16;
00608         uint32_t ui32;
00609         int32_t i32;
00610         uint64_t ui64;
00611         int64_t i64;
00612         float f;
00613         double d;
00614         tek_sa_datetime dt;
00615         struct tek_sa_string s;
00616         struct tek_sa_guid guid;
00617         struct tek_sa_byte_string bs;
00618         struct tek_sa_variant_array array;
00619         struct tek_sa_variant_matrix matrix;
00620         struct tek_sa_complex_data complex;
00621         struct tek_sa_complex_data_array complex_array;
00622         struct tek_sa_complex_data_matrix complex_matrix;
00623     } data;
00624 };
00625 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant, type, 0, 0);
00626 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant, data, 8, 8);
00627
00629 typedef struct tek_sa_variant tek_sa_field_value;
00630
00631 /*****
00632  * Type definitions from data client to TEK
00633  *****/
00634
00638 struct tek_sa_struct_field_type_definition {
00640     char* name;
00641
00643     tek_sa_type_handle_or_type_enum type;
00644 };
00645 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_field_type_definition, name, 0, 0);
00646 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_field_type_definition, type, 8, 8);
00647
00651 struct tek_sa_struct_definition {
00653     char* name;
00654
00656     struct tek_sa_struct_field_type_definition* items;
00657
00659     size_t item_count;
00660 };
00661 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, name, 0, 0);
00662 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, items, 4, 8);
00663 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, item_count, 8, 16);
00664
00669 struct tek_sa_enum_item_definition {
00671     char* name;
00672
00674     int32_t value;
00675 };
00676 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_item_definition, name, 0, 0);
00677 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_item_definition, value, 4, 8);
00678
00682 struct tek_sa_enum_definition {
00684     char* name;
00685
00687     struct tek_sa_enum_item_definition* items;
00688
00690     size_t item_count;
00691 };

```

```

00692 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, name, 0, 0);
00693 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, items, 4, 8);
00694 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, item_count, 8, 16);
00695
00701 struct tek_sa_method_argument_description {
00702     char const* name;
00703
00704     enum tek_sa_variant_type type;
00705 };
00706 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_method_argument_description, name, 0, 0);
00707 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_method_argument_description, type, 4, 8);
00708
00711 /*****
00712  * Handles and structures for data exchange
00713  *****/
00714
00716 enum tek_sa_field_attributes {
00717     TEK_SA_FIELD_ATTRIBUTES_WRITABLE = 0x1,
00718
00719     TEK_SA_FIELD_ATTRIBUTES_READABLE = 0x2,
00720
00721     TEK_SA_FIELD_ATTRIBUTES_SUBSCRIBABLE = 0x4,
00722 };
00723
00726 typedef uint32_t tek_sa_field_handle;
00727
00731 typedef uint32_t tek_sa_event_handle;
00732
00734 typedef uint32_t tek_sa_alarm_handle;
00735
00737 typedef uint32_t tek_sa_method_handle;
00738
00744 #define TEK_SA_FIELD_HANDLE_INVALID 0
00745
00747 #define TEK_SA_EVENT_HANDLE_INVALID 0
00748
00750 #define TEK_SA_ALARM_HANDLE_INVALID 0
00751
00753 #define TEK_SA_METHOD_HANDLE_INVALID 0
00754
00768 typedef int TEK_SA_RESULT;
00769
00771 #define TEK_SA_ERR_SUCCESS 0
00772
00782 #define TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE 10
00783
00788 #define TEK_SA_ERR_OUT_OF_MEMORY 11
00789
00791 #define TEK_SA_ERR_INVALID_PARAMETER 12
00792
00803 #define TEK_SA_ERR_RETRY_LATER 0xffffffff
00804
00806 #define TEK_SA_READ_RESULT_STATUS_OK 0
00807
00809 #define TEK_SA_READ_RESULT_STATUS_NOK 1
00810
00812 #define TEK_SA_READ_RESULT_STATUS_TIMEOUT 2
00813
00816 #define TEK_SA_READ_RESULT_STATUS_INVALID_HANDLE 3
00817
00824 #define TEK_SA_BLOCK_TRANSFER_END_OF_FILE 26
00825
00833 #define TEK_SA_BLOCK_TRANSFER_ABORT 24
00836 /*****
00837  * Request and response structures
00838  *****/
00839
00841 struct tek_sa_field_write_request {
00842     tek_sa_field_handle handle;
00843
00844     tek_sa_field_value value;
00845 };
00846 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_field_write_request, handle, 0, 0);
00847 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_field_write_request, value, 8, 8);
00848
00853 struct tek_sa_write_result {
00854     TEK_SA_RESULT status;
00855
00856     tek_sa_field_handle handle;
00857 };
00858 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_write_result, status, 0, 0);
00859 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_write_result, handle, 4, 4);
00860
00865 struct tek_sa_read_result {
00866     TEK_SA_RESULT status;
00867
00868     tek_sa_field_handle handle;
00869 };

```

```

00871
00878     tek_sa_field_value value;
00879 };
00880 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, status, 0, 0);
00881 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, handle, 4, 4);
00882 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, value, 8, 8);
00883
00885 struct tek_sa_event_parameter {
00887     char const* name;
00888
00890     tek_sa_field_value value;
00891 };
00892 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_event_parameter, name, 0, 0);
00893 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_event_parameter, value, 8, 8);
00894
00897 struct tek_sa_dc_event {
00904     tek_sa_datetime timestamp;
00905
00920     int16_t severity;
00921
00928     tek_sa_event_handle event_type;
00929
00936     tek_sa_field_handle source;
00937
00939     size_t number_of_parameters;
00940
00942     struct tek_sa_event_parameter* parameters;
00943 };
00944 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, timestamp, 0, 0);
00945 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, severity, 8, 8);
00946 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, event_type, 12, 12);
00947 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, source, 16, 16);
00948 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, number_of_parameters, 20, 24);
00949 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, parameters, 24, 32);
00950
00955 enum tek_sa_log_level_t {
00956     TEK_SA_LOG_LEVEL_TRACE = 0x0,
00957     TEK_SA_LOG_LEVEL_DEBUG = 0x1,
00958     TEK_SA_LOG_LEVEL_INFO = 0x2,
00959     TEK_SA_LOG_LEVEL_WARNING = 0x3,
00960     TEK_SA_LOG_LEVEL_ERROR = 0x4,
00961     TEK_SA_LOG_LEVEL_CRITICAL = 0x5,
00962 };
00963
00970 /*****
00971  * Data client capabilities
00972  *****/
00973
00978 enum tek_sa_threading_model {
00983     TEK_SA_THREADING_MODEL_SAME_THREAD = 0x0,
00984
00989     TEK_SA_THREADING_MODEL_SEQUENTIAL = 0x1,
00990
00997     TEK_SA_THREADING_MODEL_PARALLEL = 0x2,
00998 };
00999
01008 struct tek_sa_data_client_capabilities {
01018     size_t number_of_inflight_calls;
01019
01024     enum tek_sa_threading_model threading_model;
01025 };
01026 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_capabilities, number_of_inflight_calls, 0, 0);
01027 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_capabilities, threading_model, 4, 8);
01028
01034 struct tek_sa_data_client {
01052     TEK_SA_RESULT (*register_features)(tek_sa_data_client_handle dc);
01053
01066     TEK_SA_RESULT (*connect)(tek_sa_data_client_handle dc);
01067
01073     void (*free)(tek_sa_data_client_handle dc);
01074
01167     TEK_SA_RESULT(*read_fields)(tek_sa_data_client_handle dc, uint64_t request_id,
01168     const tek_sa_field_handle items_to_read[], size_t number_of_items,
01169     bool do_not_block);
01170
01189     TEK_SA_RESULT(*write_fields)(tek_sa_data_client_handle dc, uint64_t request_id,
01190     const struct tek_sa_field_write_request items_to_write[],
01191     size_t number_of_items, bool do_not_block);
01192
01214     TEK_SA_RESULT(*block_read)(const tek_sa_data_client_handle dc, uint64_t request_id,
01215     const char* filepath, uint64_t offset, int64_t length, bool do_not_block,
01216     int64_t* filesize);
01217
01237     TEK_SA_RESULT(*block_write)(const tek_sa_data_client_handle dc, uint64_t request_id,
01238     const char* filepath, uint64_t offset, int64_t length, bool do_not_block);
01239
01257     TEK_SA_RESULT(*subscribe)(tek_sa_data_client_handle dc, const tek_sa_field_handle

```



```

    items_to_subscribe[],
01258     size_t number_of_items);
01259
01270 TEK_SA_RESULT(*unsubscribe)(tek_sa_data_client_handle dc,
01271     const tek_sa_field_handle items_to_unsubscribe[], size_t number_of_items);
01272
01296 TEK_SA_RESULT(*invoke)(const tek_sa_data_client_handle dc, const tek_sa_method_handle method,
01297     uint64_t request_id, const tek_sa_field_value parameters[],
01298     const size_t number_of_parameters);
01299
01315 void (*acknowledge_alarm)(tek_sa_data_client_handle dc,
01316     const tek_sa_alarm_handle alarm);
01317
01327 tek_sa_data_client_handle handle;
01328
01330 };
01331 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, register_features, 0, 0);
01332 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, connect, 4, 8);
01333 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, free, 8, 16);
01334 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, read_fields, 12, 24);
01335 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, write_fields, 16, 32);
01336 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, block_read, 20, 40);
01337 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, block_write, 24, 48);
01338 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, subscribe, 28, 56);
01339 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, unsubscribe, 32, 64);
01340 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, invoke, 36, 72);
01341 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, acknowledge_alarm, 40, 80);
01342 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, handle, 44, 88);
01343
01351 struct tek_sa_data_client_plugin {
01352     void* plugin_context;
01353
01374 TEK_SA_RESULT(*data_client_new)(void* plugin_context, const struct tek_sa_configuration* config,
01375     struct tek_sa_data_client* created_client,
01376     struct tek_sa_data_client_capabilities* capabilities);
01377
01381 void (*free_context)(void* plugin_context);
01382 };
01383 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, plugin_context, 0, 0);
01384 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, data_client_new, 4, 8);
01385 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, free_context, 8, 16);
01386
01399 struct tek_sa_transformation_engine {
01413     tek_sa_field_handle (*register_field)(tek_sa_data_client_handle dc,
01414         const char* name,
01415         enum tek_sa_field_attributes attributes,
01416         enum tek_sa_variant_type type);
01417
01431     tek_sa_method_handle (*register_method)(
01432         tek_sa_data_client_handle dc, const char* name,
01433         struct tek_sa_method_argument_description input_parameter[],
01434         size_t number_of_input_parameters,
01435         struct tek_sa_method_argument_description output_parameter[],
01436         size_t number_of_output_parameters);
01437
01450     tek_sa_event_handle (*register_event)(tek_sa_data_client_handle dc,
01451         const char* name);
01452
01466     tek_sa_alarm_handle (*register_alarm)(tek_sa_data_client_handle dc,
01467         const char* name,
01468         const int16_t severity,
01469         const tek_sa_field_handle source);
01470
01486     TEK_SA_RESULT(*register_enum_type)(tek_sa_data_client_handle dc,
01487         struct tek_sa_enum_definition const* type_definition,
01488         tek_sa_type_handle* result);
01489
01499     TEK_SA_RESULT(*register_struct_type)(tek_sa_data_client_handle dc,
01500         struct tek_sa_struct_definition const* type_definition,
01501         tek_sa_type_handle* result);
01502
01519     TEK_SA_RESULT(*post_event)(tek_sa_data_client_handle dc, struct tek_sa_dc_event const* event);
01520
01532     TEK_SA_RESULT(*set_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm);
01533
01542     TEK_SA_RESULT(*reset_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm);
01543
01566     void (*log)(tek_sa_data_client_handle source, enum tek_sa_log_level_t lvl,
01567         const char* format, va_list args);
01568
01582     tek_sa_event_handle (*get_global_event)(const char* name);
01583
01591     void (*update_capabilities)(
01592         tek_sa_data_client_handle dc,
01593         struct tek_sa_data_client_capabilities const* capabilities);
01594
01614     void (*read_progress)(tek_sa_data_client_handle dc, uint64_t request_id,

```

```

01615         uint64_t progress);
01616
01638 void (*read_result) (tek_sa_data_client_handle dc, uint64_t request_id,
01639                     TEK_SA_RESULT result,
01640                     const struct tek_sa_read_result results[],
01641                     size_t number_of_results);
01642
01650 void (*notify_change) (tek_sa_data_client_handle dc,
01651                       const struct tek_sa_read_result changes[],
01652                       size_t number_of_changes);
01653
01664 void (*write_result) (tek_sa_data_client_handle dc, uint64_t request_id,
01665                     TEK_SA_RESULT result,
01666                     const struct tek_sa_write_result results[],
01667                     size_t number_of_results);
01668
01680 void (*call_method_result) (tek_sa_data_client_handle dc, uint64_t request_id,
01681                           TEK_SA_RESULT result,
01682                           const tek_sa_field_value results[],
01683                           size_t number_of_results);
01684
01701 TEK_SA_RESULT(*block_read_data) (tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT
result,
01702 unsigned char buffer[], size_t buffer_length);
01703
01718 TEK_SA_RESULT(*block_write_data) (tek_sa_data_client_handle dc, uint64_t request_id, unsigned char
buffer[],
01719 size_t buffer_length, size_t* bytes_written);
01720
01729 void (*block_write_result) (tek_sa_data_client_handle dc, uint64_t request_id,
01730                             TEK_SA_RESULT result);
01731
01733 };
01734 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_field, 0, 0);
01735 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_method, 4, 8);
01736 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_event, 8, 16);
01737 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_alarm, 12, 24);
01738 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_enum_type, 16, 32);
01739 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_struct_type, 20, 40);
01740 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, post_event, 24, 48);
01741 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, set_alarm, 28, 56);
01742 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, reset_alarm, 32, 64);
01743 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, log, 36, 72);
01744 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, get_global_event, 40, 80);
01745 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, update_capabilities, 44, 88);
01746 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, read_progress, 48, 96);
01747 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, read_result, 52, 104);
01748 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, notify_change, 56, 112);
01749 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, write_result, 60, 120);
01750 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, call_method_result, 64, 128);
01751 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_read_data, 68, 136);
01752 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_write_data, 72, 144);
01753 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_write_result, 76, 152);
01754
01769 typedef TEK_SA_RESULT (*tek_sa_load_plugin_fn) (
01770     struct tek_sa_transformation_engine* api,
01771     const struct tek_sa_configuration* plugin_configuration,
01772     struct tek_sa_data_client_plugin* plugin);
01773
01774 #ifdef TEK_SA_DATA_CLIENT_IMPL
01775
01776 #ifdef _WIN32
01777 #define TEK_SA_API_EXPORT __declspec(dllexport) __stdcall
01778 #else
01779 #define TEK_SA_API_EXPORT __attribute__((__visibility__("default")))
01780 #endif
01781
01785 TEK_SA_RESULT TEK_SA_API_EXPORT
01786     load_plugin(struct tek_sa_transformation_engine* api,
01787               const struct tek_sa_configuration* plugin_configuration,
01788               struct tek_sa_data_client_plugin* plugin);
01789
01790 #endif
01791
01792 #ifdef __cplusplus
01793 }
01794 #endif
01795
01796 #undef TEK_SA_STRUCT_ALIGN_SELECT
01797 #undef TEK_SA_VERIFY_STRUCT_OFFSET
01798
01799 #endif /* TEK_SOUTH_API_H */

```

# Index

- acknowledge\_alarm
  - tek\_sa\_data\_client, [44](#)
- block\_read
  - tek\_sa\_data\_client, [41](#)
- block\_read\_data
  - tek\_sa\_transformation\_engine, [55](#)
- block\_write
  - tek\_sa\_data\_client, [42](#)
- block\_write\_data
  - tek\_sa\_transformation\_engine, [55](#)
- block\_write\_result
  - tek\_sa\_transformation\_engine, [56](#)
- call\_method\_result
  - tek\_sa\_transformation\_engine, [55](#)
- Common Definitions, [18](#)
  - tek\_sa\_alarm\_handle, [34](#)
  - TEK\_SA\_ALARM\_HANDLE\_INVALID, [30](#)
  - TEK\_SA\_BLOCK\_TRANSFER\_ABORT, [32](#)
  - TEK\_SA\_BLOCK\_TRANSFER\_END\_OF\_FILE, [32](#)
  - tek\_sa\_datetime, [33](#)
  - TEK\_SA\_ERR\_INVALID\_PARAMETER, [31](#)
  - TEK\_SA\_ERR\_NON\_BLOCKING\_IMPOSSIBLE, [30](#)
  - TEK\_SA\_ERR\_OUT\_OF\_MEMORY, [31](#)
  - TEK\_SA\_ERR\_RETRY\_LATER, [31](#)
  - TEK\_SA\_ERR\_SUCCESS, [30](#)
  - tek\_sa\_event\_handle, [34](#)
  - TEK\_SA\_EVENT\_HANDLE\_INVALID, [30](#)
  - tek\_sa\_field\_attributes, [36](#)
  - TEK\_SA\_FIELD\_ATTRIBUTES\_READABLE, [36](#)
  - TEK\_SA\_FIELD\_ATTRIBUTES\_SUBSCRIBABLE, [36](#)
  - TEK\_SA\_FIELD\_ATTRIBUTES\_WRITABLE, [36](#)
  - tek\_sa\_field\_handle, [34](#)
  - TEK\_SA\_FIELD\_HANDLE\_INVALID, [29](#)
  - tek\_sa\_field\_value, [33](#)
  - TEK\_SA\_LOG\_LEVEL\_CRITICAL, [36](#)
  - TEK\_SA\_LOG\_LEVEL\_DEBUG, [36](#)
  - TEK\_SA\_LOG\_LEVEL\_ERROR, [36](#)
  - TEK\_SA\_LOG\_LEVEL\_INFO, [36](#)
  - tek\_sa\_log\_level\_t, [36](#)
  - TEK\_SA\_LOG\_LEVEL\_TRACE, [36](#)
  - TEK\_SA\_LOG\_LEVEL\_WARNING, [36](#)
  - tek\_sa\_method\_handle, [34](#)
  - TEK\_SA\_METHOD\_HANDLE\_INVALID, [30](#)
  - TEK\_SA\_READ\_RESULT\_STATUS\_INVALID\_HANDLE, [32](#)
  - TEK\_SA\_READ\_RESULT\_STATUS\_NOK, [32](#)
  - TEK\_SA\_READ\_RESULT\_STATUS\_OK, [31](#)
  - TEK\_SA\_READ\_RESULT\_STATUS\_TIMEOUT, [32](#)
  - TEK\_SA\_RESULT, [34](#)
  - tek\_sa\_type\_handle, [33](#)
  - tek\_sa\_type\_handle\_or\_type\_enum, [33](#)
  - tek\_sa\_variant\_type, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_BOOL, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_BYTE\_STRING, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_COMPLEX, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_DATETIME, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_DOUBLE, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_FLAG\_ARRAY, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_FLAG\_MATRIX, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_FLOAT, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_GUID, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_INT16\_T, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_INT32\_T, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_INT64\_T, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_INT8\_T, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_NULL, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_STRING, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_UINT16\_T, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_UINT32\_T, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_UINT64\_T, [35](#)
  - TEK\_SA\_VARIANT\_TYPE\_UINT8\_T, [35](#)
- connect
  - tek\_sa\_data\_client, [38](#)
- Data Client, [15](#)
  - tek\_sa\_data\_client\_handle, [17](#)
  - tek\_sa\_load\_plugin\_fn, [17](#)
  - tek\_sa\_threading\_model, [17](#)
  - TEK\_SA\_THREADING\_MODEL\_PARALLEL, [18](#)
  - TEK\_SA\_THREADING\_MODEL\_SAME\_THREAD, [18](#)
  - TEK\_SA\_THREADING\_MODEL\_SEQUENTIAL, [18](#)
- data\_client\_new
  - tek\_sa\_data\_client\_plugin, [45](#)
- free
  - tek\_sa\_data\_client, [39](#)
- free\_context
  - tek\_sa\_data\_client\_plugin, [46](#)
- get\_global\_event
  - tek\_sa\_transformation\_engine, [52](#)
- handle

- tek\_sa\_data\_client, 44
- include/south\_api.h, 57, 61
- invoke
  - tek\_sa\_data\_client, 43
- log
  - tek\_sa\_transformation\_engine, 52
- notify\_change
  - tek\_sa\_transformation\_engine, 54
- plugin\_context
  - tek\_sa\_data\_client\_plugin, 45
- post\_event
  - tek\_sa\_transformation\_engine, 50
- read\_fields
  - tek\_sa\_data\_client, 39
- read\_progress
  - tek\_sa\_transformation\_engine, 53
- read\_result
  - tek\_sa\_transformation\_engine, 53
- register\_alarm
  - tek\_sa\_transformation\_engine, 49
- register\_enum\_type
  - tek\_sa\_transformation\_engine, 49
- register\_event
  - tek\_sa\_transformation\_engine, 49
- register\_features
  - tek\_sa\_data\_client, 38
- register\_field
  - tek\_sa\_transformation\_engine, 48
- register\_method
  - tek\_sa\_transformation\_engine, 48
- register\_struct\_type
  - tek\_sa\_transformation\_engine, 50
- reset\_alarm
  - tek\_sa\_transformation\_engine, 51
- set\_alarm
  - tek\_sa\_transformation\_engine, 51
- south\_api.h
  - TEK\_SA\_API\_VERSION, 61
  - TEK\_SA\_API\_VERSION\_MAJOR, 60
  - TEK\_SA\_API\_VERSION\_MINOR, 61
  - TEK\_SA\_API\_VERSION\_PATCH, 61
- subscribe
  - tek\_sa\_data\_client, 42
- tek\_sa\_additional\_file, 21
- tek\_sa\_alarm\_handle
  - Common Definitions, 34
- TEK\_SA\_ALARM\_HANDLE\_INVALID
  - Common Definitions, 30
- TEK\_SA\_API\_VERSION
  - south\_api.h, 61
- TEK\_SA\_API\_VERSION\_MAJOR
  - south\_api.h, 60
- TEK\_SA\_API\_VERSION\_MINOR
  - south\_api.h, 61
- TEK\_SA\_API\_VERSION\_PATCH
  - south\_api.h, 61
- TEK\_SA\_BLOCK\_TRANSFER\_ABORT
  - Common Definitions, 32
- TEK\_SA\_BLOCK\_TRANSFER\_END\_OF\_FILE
  - Common Definitions, 32
- tek\_sa\_byte\_string, 22
- tek\_sa\_complex\_data, 22
- tek\_sa\_complex\_data\_array, 23
- tek\_sa\_complex\_data\_array\_item, 23
- tek\_sa\_complex\_data\_matrix, 23
- tek\_sa\_configuration, 21
- tek\_sa\_data\_client, 37
  - acknowledge\_alarm, 44
  - block\_read, 41
  - block\_write, 42
  - connect, 38
  - free, 39
  - handle, 44
  - invoke, 43
  - read\_fields, 39
  - register\_features, 38
  - subscribe, 42
  - unsubscribe, 43
  - write\_fields, 40
- tek\_sa\_data\_client\_capabilities, 16
- tek\_sa\_data\_client\_handle
  - Data Client, 17
- tek\_sa\_data\_client\_plugin, 44
  - data\_client\_new, 45
  - free\_context, 46
  - plugin\_context, 45
- tek\_sa\_datetime
  - Common Definitions, 33
- tek\_sa\_dc\_event, 27
- tek\_sa\_enum\_definition, 26
- tek\_sa\_enum\_item\_definition, 26
- TEK\_SA\_ERR\_INVALID\_PARAMETER
  - Common Definitions, 31
- TEK\_SA\_ERR\_NON\_BLOCKING\_IMPOSSIBLE
  - Common Definitions, 30
- TEK\_SA\_ERR\_OUT\_OF\_MEMORY
  - Common Definitions, 31
- TEK\_SA\_ERR\_RETRY\_LATER
  - Common Definitions, 31
- TEK\_SA\_ERR\_SUCCESS
  - Common Definitions, 30
- tek\_sa\_event\_handle
  - Common Definitions, 34
- TEK\_SA\_EVENT\_HANDLE\_INVALID
  - Common Definitions, 30
- tek\_sa\_event\_parameter, 27
- tek\_sa\_field\_attributes
  - Common Definitions, 36
- TEK\_SA\_FIELD\_ATTRIBUTES\_READABLE
  - Common Definitions, 36
- TEK\_SA\_FIELD\_ATTRIBUTES\_SUBSCRIBABLE

- Common Definitions, [36](#)
- TEK\_SA\_FIELD\_ATTRIBUTES\_WRITABLE
  - Common Definitions, [36](#)
- tek\_sa\_field\_handle
  - Common Definitions, [34](#)
- TEK\_SA\_FIELD\_HANDLE\_INVALID
  - Common Definitions, [29](#)
- tek\_sa\_field\_value
  - Common Definitions, [33](#)
- tek\_sa\_field\_write\_request, [26](#)
- tek\_sa\_guid, [21](#)
- tek\_sa\_load\_plugin\_fn
  - Data Client, [17](#)
- TEK\_SA\_LOG\_LEVEL\_CRITICAL
  - Common Definitions, [36](#)
- TEK\_SA\_LOG\_LEVEL\_DEBUG
  - Common Definitions, [36](#)
- TEK\_SA\_LOG\_LEVEL\_ERROR
  - Common Definitions, [36](#)
- TEK\_SA\_LOG\_LEVEL\_INFO
  - Common Definitions, [36](#)
- tek\_sa\_log\_level\_t
  - Common Definitions, [36](#)
- TEK\_SA\_LOG\_LEVEL\_TRACE
  - Common Definitions, [36](#)
- TEK\_SA\_LOG\_LEVEL\_WARNING
  - Common Definitions, [36](#)
- tek\_sa\_method\_argument\_description, [26](#)
- tek\_sa\_method\_handle
  - Common Definitions, [34](#)
- TEK\_SA\_METHOD\_HANDLE\_INVALID
  - Common Definitions, [30](#)
- tek\_sa\_read\_result, [27](#)
- TEK\_SA\_READ\_RESULT\_STATUS\_INVALID\_HANDLE
  - Common Definitions, [32](#)
- TEK\_SA\_READ\_RESULT\_STATUS\_NOK
  - Common Definitions, [32](#)
- TEK\_SA\_READ\_RESULT\_STATUS\_OK
  - Common Definitions, [31](#)
- TEK\_SA\_READ\_RESULT\_STATUS\_TIMEOUT
  - Common Definitions, [32](#)
- TEK\_SA\_RESULT
  - Common Definitions, [34](#)
- tek\_sa\_string, [22](#)
- tek\_sa\_struct\_definition, [25](#)
- tek\_sa\_struct\_field\_type\_definition, [25](#)
- tek\_sa\_threading\_model
  - Data Client, [17](#)
- TEK\_SA\_THREADING\_MODEL\_PARALLEL
  - Data Client, [18](#)
- TEK\_SA\_THREADING\_MODEL\_SAME\_THREAD
  - Data Client, [18](#)
- TEK\_SA\_THREADING\_MODEL\_SEQUENTIAL
  - Data Client, [18](#)
- tek\_sa\_transformation\_engine, [46](#)
  - block\_read\_data, [55](#)
  - block\_write\_data, [55](#)
  - block\_write\_result, [56](#)
- call\_method\_result, [55](#)
- get\_global\_event, [52](#)
- log, [52](#)
- notify\_change, [54](#)
- post\_event, [50](#)
- read\_progress, [53](#)
- read\_result, [53](#)
- register\_alarm, [49](#)
- register\_enum\_type, [49](#)
- register\_event, [49](#)
- register\_field, [48](#)
- register\_method, [48](#)
- register\_struct\_type, [50](#)
- reset\_alarm, [51](#)
- set\_alarm, [51](#)
- update\_capabilities, [52](#)
- write\_result, [54](#)
- tek\_sa\_type\_handle
  - Common Definitions, [33](#)
- tek\_sa\_type\_handle\_or\_type\_enum
  - Common Definitions, [33](#)
- tek\_sa\_variant, [25](#)
- tek\_sa\_variant.data, [29](#)
- tek\_sa\_variant\_array, [24](#)
- tek\_sa\_variant\_array.data, [28](#)
- tek\_sa\_variant\_matrix, [24](#)
- tek\_sa\_variant\_type
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_BOOL
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_BYTE\_STRING
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_COMPLEX
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_DATETIME
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_DOUBLE
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_FLAG\_ARRAY
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_FLAG\_MATRIX
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_FLOAT
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_GUID
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_INT16\_T
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_INT32\_T
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_INT64\_T
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_INT8\_T
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_NULL
  - Common Definitions, [35](#)
- TEK\_SA\_VARIANT\_TYPE\_STRING
  - Common Definitions, [35](#)

TEK\_SA\_VARIANT\_TYPE\_UINT16\_T  
    Common Definitions, [35](#)  
TEK\_SA\_VARIANT\_TYPE\_UINT32\_T  
    Common Definitions, [35](#)  
TEK\_SA\_VARIANT\_TYPE\_UINT64\_T  
    Common Definitions, [35](#)  
TEK\_SA\_VARIANT\_TYPE\_UINT8\_T  
    Common Definitions, [35](#)  
tek\_sa\_write\_result, [27](#)  
Transformation Engine, [15](#)  
  
unsubscribe  
    tek\_sa\_data\_client, [43](#)  
update\_capabilities  
    tek\_sa\_transformation\_engine, [52](#)  
  
write\_fields  
    tek\_sa\_data\_client, [40](#)  
write\_result  
    tek\_sa\_transformation\_engine, [54](#)