



Forschungsinstitut

umati Transformation Engine - API documentation

(Release Candidate, 2021-10-01)

1 Introduction	1
1.1 Recommended Reading	1
2 Initialization of a data client plugin	3
3 Known issues	5
3.1 API definition issues	5
3.2 Documentation/Style issues	5
4 Todo List	7
5 Module Index	9
5.1 Modules	9
6 Data Structure Index	11
6.1 Data Structures	11
7 File Index	13
7.1 File List	13
8 Module Documentation	15
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
9 Data Structure Documentation	37
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
10 File Documentation	57
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
Index	69

Chapter 1

Introduction

The [VDW-Forschungsinstitut e.V.](#) is currently working with partners and its members to create a specification of a TransformationEngine.

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

Application Warning Notice

This DRAFT with date of issue 2021-10-01 is being submitted to the public for review and comment. Because the final API Specification may differ from this version, the application of this draft is subject to special agreement.

Comments are requested:

- preferably as a file by e-mail to g.goerisch@vdw.de
- or in paper form to VDW-Forschungsinstitut e.V., Lyoner Straße 18, 60528 Frankfurt

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](#).

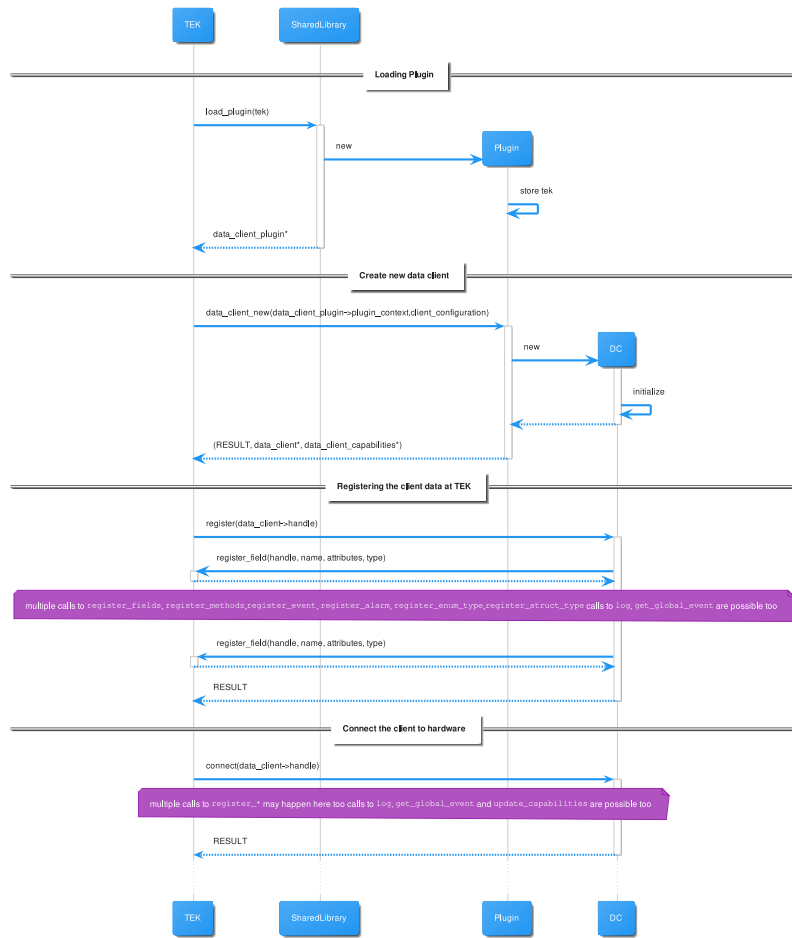
After loading the shared library the TEK calls the main initialization function with the fixed name `load_plugin` and a signature of [tek_sa_load_plugin_fn](#) . This function creates a new singleton instance of [tek_sa_data_client_plugin](#) and is expected to save the given TEK api struct.

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] mkd docs/doxybook2 output can not handle union

Todo [C, MIG] mkd docs/doxybook2 output can not handle typedefs

Todo [C, MIG] mkd docs/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:

tek_sa_data_client	
The interface of one instance of a data client	37
tek_sa_data_client_plugin	
Interface of the data client plugin	44
tek_sa_transformation_engine	
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 [1023](#) of file [south_api.h](#).

Data Fields

size_t	number_of_inflight_calls	<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>Remarks</p> <p>This information may be dependent on the physical device and therefore available only after the connection was established.</p>
enum tek_sa_threading_model	threading_model	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 230 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 1784 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>Remarks</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 993 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 240 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 255 of file [south_api.h](#).

Data Fields

char *	config	The configuration file as UTF-8 encoded JSON string
struct tek_sa_additional_file *	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 298 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 323 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 341 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 366 of file [south_api.h](#).

Data Fields

tek_sa_type_handle	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 https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.6/ .

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 396 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 tek_sa_complex_data::data

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 425 of file [south_api.h](#).

Data Fields

tek_sa_type_handle	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 tek_sa_complex_data_array_item *	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 450 of file [south_api.h](#).

Data Fields

tek_sa_type_handle	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: https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.2/#5.2.2.16
struct tek_sa_complex_data_array_item *	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 553 of file [south_api.h](#).

Data Fields

int32_t	length	The number of elements in the array.
union tek_sa_variant_array.data	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 581 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: https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.2/#5.2.2.16
struct tek_sa_variant_array	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 608 of file [south_api.h](#).

Data Fields

uint8_t	type	The type of the value. Must be one of the values described in tek_sa_variant_type .
union tek_sa_variant.data	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 653 of file [south_api.h](#).

Data Fields

char *	name	The name of the data field.
tek_sa_type_handle_or_type_enum	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 666 of file [south_api.h](#).

Data Fields

char *	name	The name of the type.
struct tek_sa_struct_field_type_definition *	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 684 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 697 of file [south_api.h](#).

Data Fields

char *	name	The name of the type.
struct tek_sa_enum_item_definition *	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 716 of file [south_api.h](#).

Data Fields

char const *	name	The name of the method parameter.
enum tek_sa_variant_type	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 856 of file [south_api.h](#).

Data Fields

tek_sa_field_handle	handle	The field handle as returned from tek_sa_transformation_engine::register_field .
tek_sa_field_value	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 868 of file [south_api.h](#).

Data Fields

TEK_SA_RESULT	status	The write operation result.
tek_sa_field_handle	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 880 of file [south_api.h](#).

Data Fields

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

8.3.2.21 struct tek_sa_event_parameter

Structure to encapsulate an event parameter.

Definition at line 900 of file [south_api.h](#).

Data Fields

<code>char const *</code>	name	The name of the parameter.
tek_sa_field_value	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 912 of file [south_api.h](#).

Data Fields

tek_sa_datetime	timestamp	<p>The Timestamp of the event.</p> <p>Remarks</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 https://reference.opcfoundation.org/v104/Core/docs/Part5/6.4.2/ 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>
tek_sa_event_handle	event_type	<p>The event type handle as returned by the call to tek_sa_transformation_engine::register_event.</p> <p>Attention</p> <p>This field must not be TEK_SA_EVENT_HANDLE_INVALID</p>
tek_sa_field_handle	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 TEK_SA_FIELD_HANDLE_INVALID.</p>
size_t	number_of_parameters	The number of event parameters.
struct tek_sa_event_parameter *	parameters	The event parameters.

8.3.2.23 union tek_sa_variant_array.data

The array values.

Definition at line 558 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	
tek_sa_datetime *	dt	
struct tek_sa_string *	s	
struct tek_sa_guid *	guid	
struct tek_sa_byte_string *	bs	

8.3.2.24 union [tek_sa_variant.data](#)

The value.

Definition at line 617 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	
tek_sa_datetime	dt	
struct tek_sa_string	s	
struct tek_sa_guid	guid	
struct tek_sa_byte_string	bs	
struct tek_sa_variant_array	array	
struct tek_sa_variant_matrix	matrix	
struct tek_sa_complex_data	complex	
struct tek_sa_complex_data_array	complex_array	
struct tek_sa_complex_data_matrix	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 759 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 762 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 765 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 768 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 786 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 797 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 803 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 806 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 818 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 821 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 824 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 827 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 831 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 839 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 848 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 282 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 288 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 357 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 644 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 743 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 746 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 749 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 752 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 783 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 tek_sa_datetime .
TEK_SA_VARIANT_TYPE_STRING	The type id of a string value. See tek_sa_string .
TEK_SA_VARIANT_TYPE_GUID	The type id of a GUID value. See tek_sa_guid .
TEK_SA_VARIANT_TYPE_BYTE_STRING	The type id of a byte string value. See tek_sa_byte_string .
TEK_SA_VARIANT_TYPE_COMPLEX	The type id of a value with a complex data type. See tek_sa_transformation_engine::register_struct_type .
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 489 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 731 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 970 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 1049 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 1067 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 1081 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 1088 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 handle of the data client as returned from tek_sa_data_client_plugin::data_client_new .
<i>request_id</i>	A unique request identifier which is created by the TEK and must be passed to call to tek_sa_transformation_engine::read_result and tek_sa_transformation_engine::read_progress .
<i>items_to_read</i>	An array of field handles which describes the values the data client should read. See also function tek_sa_transformation_engine::register_field .
<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`

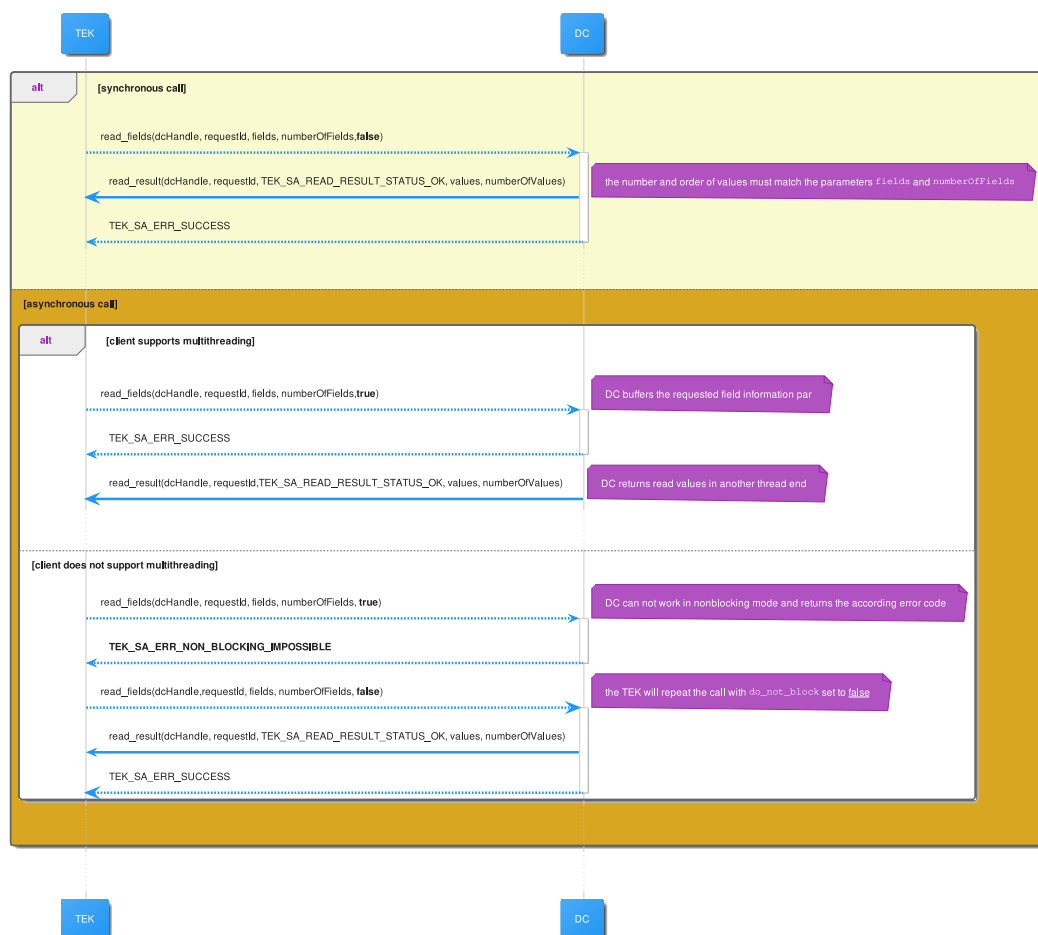


Figure 9.1 Possible call sequences

Definition at line 1182 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 handle of the data client as returned from tek_sa_data_client_plugin::data_client_new .
<i>request_id</i>	A unique request identifier which is created by the TEK and must be passed to call to tek_sa_transformation_engine::write_result .
<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 Usage in read_fields

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

Definition at line 1204 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 Usage in read_fields
<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 1229 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 handle of the data client as returned from tek_sa_data_client_plugin::data_client_new .
<i>request_id</i>	A unique request identifier which is created by the TEK and must be passed to call to tek_sa_transformation_engine::block_write_result and tek_sa_transformation_engine::block_write_data .
<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 Usage in read_fields

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 1252 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 handle of the data client as returned from tek_sa_data_client_plugin::data_client_new .
<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 1272 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 handle of the data client as returned from tek_sa_data_client_plugin::data_client_new .
<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 1285 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 handle of the data client as returned from tek_sa_data_client_plugin::data_client_new .
<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 tek_sa_transformation_engine::block_write_result and tek_sa_transformation_engine::block_write_data .
<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 1311 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 handle of the data client as returned from tek_sa_data_client_plugin::data_client_new .
<i>alarm</i>	An alarm handle which is returned from the method tek_sa_transformation_engine::register_alarm .

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 1330 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 1342 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 1366 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 1371 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 1389 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 1396 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 1414 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 1428 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.
tek_sa_method_argument_description	The description of the method input arguments.
<i>number_of_input_parameters</i>	The number of input parameters.
tek_sa_method_argument_description	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 1446 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 1465 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 1481 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.
tek_sa_enum_definition	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 1501 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.
tek_sa_struct_definition	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 1514 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 1534 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 1547 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 1557 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 1581 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 1597 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 1606 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 1629 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 1653 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 1665 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 1679 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 1695 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 1716 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 1733 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 1744 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
00230 typedef void* tek_sa_data_client_handle;
00231
00232 /*****
00233  * Configuration structures
00234  *****/
00235
00240 struct tek_sa_additional_file {
00242     char* name;
00243
00245     char* content;
00246 };
00247
00248 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_additional_file, name, 0, 0);
00249 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_additional_file, content, 4, 8);
00250
00255 struct tek_sa_configuration {
00257     char* config;
00258
00260     struct tek_sa_additional_file* additional_files;
00261
00263     size_t additional_files_count;
00264 };
00265
00266 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, config, 0, 0);
00267 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, additional_files, 4, 8);
00268 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, additional_files_count, 8, 16);
00269
00270 /*****
00271  * Built-in type definitions and variant
00272  *****/
00273
00282 typedef int64_t tek_sa_type_handle;
00283
00288 typedef int64_t tek_sa_type_handle_or_type_enum;
00289
00298 struct tek_sa_guid {
00300     uint32_t data1;
00301
00303     uint16_t data2;
00304
00306     uint16_t data3;
00307
00309     uint8_t data4[8];
00310 };
00311 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data1, 0, 0);
00312 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data2, 4, 4);
00313 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data3, 6, 6);
00314 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data4, 8, 8);
00315
00323 struct tek_sa_byte_string {
00325     int32_t length;
00326
00328     unsigned char* data;
00329 };
00330 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_byte_string, length, 0, 0);
00331 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_byte_string, data, 4, 8);
00332
00341 struct tek_sa_string {
00343     int32_t length;
00344
00346     unsigned char* data;
00347 };
00348 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_string, length, 0, 0);
00349 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_string, data, 4, 8);
00350
00357 typedef int64_t tek_sa_datetime;
00358
00366 struct tek_sa_complex_data {
00368     tek_sa_type_handle type;
00369
00376     uint32_t data_length;
00377
00384     unsigned char* data;
00385 };
00386 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, type, 0, 0);
00387 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, data_length, 8, 8);
00388 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, data, 12, 16);
00389
00396 struct tek_sa_complex_data_array_item {
00404     uint32_t data_length;
00405

```

```

00411 unsigned char* data;
00412 };
00413 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array_item, data_length, 0, 0);
00414 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array_item, data, 4, 8);
00415
00425 struct tek_sa_complex_data_array {
00427     tek_sa_type_handle type;
00428
00430     size_t number_of_items;
00431
00434     struct tek_sa_complex_data_array_item* data;
00435 };
00436
00437 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, type, 0, 0);
00438 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, number_of_items, 8, 8);
00439 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, data, 12, 16);
00440
00450 struct tek_sa_complex_data_matrix {
00452     tek_sa_type_handle type;
00453
00455     int32_t dimension_length;
00456
00471     int32_t* dimensions;
00472
00475     struct tek_sa_complex_data_array_item* data;
00476 };
00477 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, type, 0, 0);
00478 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, dimension_length, 8, 8);
00479 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, dimensions, 12, 16);
00480 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, data, 16, 24);
00481
00489 enum tek_sa_variant_type {
00491     TEK_SA_VARIANT_TYPE_NULL = 0x0,
00492
00494     TEK_SA_VARIANT_TYPE_BOOL = 0x1,
00495
00497     TEK_SA_VARIANT_TYPE_UINT8_T = 0x2,
00498
00500     TEK_SA_VARIANT_TYPE_INT8_T = 0x3,
00501
00503     TEK_SA_VARIANT_TYPE_UINT16_T = 0x4,
00504
00506     TEK_SA_VARIANT_TYPE_INT16_T = 0x5,
00507
00509     TEK_SA_VARIANT_TYPE_UINT32_T = 0x6,
00510
00512     TEK_SA_VARIANT_TYPE_INT32_T = 0x7,
00513
00515     TEK_SA_VARIANT_TYPE_UINT64_T = 0x8,
00516
00518     TEK_SA_VARIANT_TYPE_INT64_T = 0x9,
00519
00521     TEK_SA_VARIANT_TYPE_FLOAT = 0xa,
00522
00524     TEK_SA_VARIANT_TYPE_DOUBLE = 0xb,
00525
00527     TEK_SA_VARIANT_TYPE_DATETIME = 0xc,
00528
00530     TEK_SA_VARIANT_TYPE_STRING = 0xd,
00531
00533     TEK_SA_VARIANT_TYPE_GUID = 0xe,
00534
00536     TEK_SA_VARIANT_TYPE_BYTE_STRING = 0xf,
00537
00540     TEK_SA_VARIANT_TYPE_COMPLEX = 0x20,
00541
00544     TEK_SA_VARIANT_TYPE_FLAG_ARRAY = 0x40,
00545
00548     TEK_SA_VARIANT_TYPE_FLAG_MATRIX = 0x80
00549 };
00550
00553 struct tek_sa_variant_array {
00555     int32_t length;
00556
00558     union {
00559         bool* b;
00560         uint8_t* ui8;
00561         int8_t* i8;
00562         uint16_t* ui16;
00563         int16_t* i16;
00564         uint32_t* ui32;
00565         int32_t* i32;
00566         uint64_t* ui64;
00567         int64_t* i64;
00568         float* f;
00569         double* d;
00570         tek_sa_datetime* dt;

```

```

00571     struct tek_sa_string* s;
00572     struct tek_sa_guid* guid;
00573     struct tek_sa_byte_string* bs;
00574 } data;
00575 };
00576 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_array, length, 0, 0);
00577 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_array, data, 4, 8);
00578
00581 struct tek_sa_variant_matrix {
00583     int32_t dimension_length;
00584
00598     int32_t* dimensions;
00599
00601     struct tek_sa_variant_array data;
00602 };
00603 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_matrix, dimension_length, 0, 0);
00604 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_matrix, dimensions, 4, 8);
00605
00608 struct tek_sa_variant {
00614     uint8_t type;
00615
00617     union {
00618         bool b;
00619         uint8_t ui8;
00620         int8_t i8;
00621         uint16_t ui16;
00622         int16_t i16;
00623         uint32_t ui32;
00624         int32_t i32;
00625         uint64_t ui64;
00626         int64_t i64;
00627         float f;
00628         double d;
00629         tek_sa_datetime dt;
00630         struct tek_sa_string s;
00631         struct tek_sa_guid guid;
00632         struct tek_sa_byte_string bs;
00633         struct tek_sa_variant_array array;
00634         struct tek_sa_variant_matrix matrix;
00635         struct tek_sa_complex_data complex;
00636         struct tek_sa_complex_data_array complex_array;
00637         struct tek_sa_complex_data_matrix complex_matrix;
00638     } data;
00639 };
00640 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant, type, 0, 0);
00641 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant, data, 8, 8);
00642
00644 typedef struct tek_sa_variant tek_sa_field_value;
00645
00646 /*****
00647  * Type definitions from data client to TEK
00648  *****/
00649
00653 struct tek_sa_struct_field_type_definition {
00655     char* name;
00656
00658     tek_sa_type_handle_or_type_enum type;
00659 };
00660 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_field_type_definition, name, 0, 0);
00661 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_field_type_definition, type, 8, 8);
00662
00666 struct tek_sa_struct_definition {
00668     char* name;
00669
00671     struct tek_sa_struct_field_type_definition* items;
00672
00674     size_t item_count;
00675 };
00676 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, name, 0, 0);
00677 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, items, 4, 8);
00678 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, item_count, 8, 16);
00679
00684 struct tek_sa_enum_item_definition {
00686     char* name;
00687
00689     int32_t value;
00690 };
00691 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_item_definition, name, 0, 0);
00692 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_item_definition, value, 4, 8);
00693
00697 struct tek_sa_enum_definition {
00699     char* name;
00700
00702     struct tek_sa_enum_item_definition* items;
00703
00705     size_t item_count;
00706 };

```

```

00707 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, name, 0, 0);
00708 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, items, 4, 8);
00709 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, item_count, 8, 16);
00710
00716 struct tek_sa_method_argument_description {
00717     char const* name;
00719
00721     enum tek_sa_variant_type type;
00722 };
00723 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_method_argument_description, name, 0, 0);
00724 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_method_argument_description, type, 4, 8);
00725
00726 /*****
00727  * Handles and structures for data exchange
00728  *****/
00729
00731 enum tek_sa_field_attributes {
00733     TEK_SA_FIELD_ATTRIBUTES_WRITABLE = 0x1,
00734
00736     TEK_SA_FIELD_ATTRIBUTES_READABLE = 0x2,
00737
00739     TEK_SA_FIELD_ATTRIBUTES_SUBSCRIBABLE = 0x4,
00740 };
00741
00743 typedef uint32_t tek_sa_field_handle;
00744
00746 typedef uint32_t tek_sa_event_handle;
00747
00749 typedef uint32_t tek_sa_alarm_handle;
00750
00752 typedef uint32_t tek_sa_method_handle;
00753
00759 #define TEK_SA_FIELD_HANDLE_INVALID 0
00760
00762 #define TEK_SA_EVENT_HANDLE_INVALID 0
00763
00765 #define TEK_SA_ALARM_HANDLE_INVALID 0
00766
00768 #define TEK_SA_METHOD_HANDLE_INVALID 0
00769
00783 typedef int TEK_SA_RESULT;
00784
00786 #define TEK_SA_ERR_SUCCESS 0
00787
00797 #define TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE 10
00798
00803 #define TEK_SA_ERR_OUT_OF_MEMORY 11
00804
00806 #define TEK_SA_ERR_INVALID_PARAMETER 12
00807
00818 #define TEK_SA_ERR_RETRY_LATER 0xffffffff
00819
00821 #define TEK_SA_READ_RESULT_STATUS_OK 0
00822
00824 #define TEK_SA_READ_RESULT_STATUS_NOK 1
00825
00827 #define TEK_SA_READ_RESULT_STATUS_TIMEOUT 2
00828
00831 #define TEK_SA_READ_RESULT_STATUS_INVALID_HANDLE 3
00832
00839 #define TEK_SA_BLOCK_TRANSFER_END_OF_FILE 26
00840
00848 #define TEK_SA_BLOCK_TRANSFER_ABORT 24
00851 /*****
00852  * Request and response structures
00853  *****/
00854
00856 struct tek_sa_field_write_request {
00859     tek_sa_field_handle handle;
00860
00862     tek_sa_field_value value;
00863 };
00864 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_field_write_request, handle, 0, 0);
00865 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_field_write_request, value, 8, 8);
00866
00868 struct tek_sa_write_result {
00870     TEK_SA_RESULT status;
00873     tek_sa_field_handle handle;
00874 };
00875 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_write_result, status, 0, 0);
00876 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_write_result, handle, 4, 4);
00877
00880 struct tek_sa_read_result {
00882     TEK_SA_RESULT status;
00883
00885     tek_sa_field_handle handle;

```

```

00886
00893     tek_sa_field_value value;
00894 };
00895 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, status, 0, 0);
00896 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, handle, 4, 4);
00897 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, value, 8, 8);
00898
00900 struct tek_sa_event_parameter {
00902     char const* name;
00903
00905     tek_sa_field_value value;
00906 };
00907 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_event_parameter, name, 0, 0);
00908 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_event_parameter, value, 8, 8);
00909
00912 struct tek_sa_dc_event {
00919     tek_sa_datetime timestamp;
00920
00935     int16_t severity;
00936
00943     tek_sa_event_handle event_type;
00944
00951     tek_sa_field_handle source;
00952
00954     size_t number_of_parameters;
00955
00957     struct tek_sa_event_parameter* parameters;
00958 };
00959 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, timestamp, 0, 0);
00960 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, severity, 8, 8);
00961 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, event_type, 12, 12);
00962 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, source, 16, 16);
00963 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, number_of_parameters, 20, 24);
00964 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, parameters, 24, 32);
00965
00970 enum tek_sa_log_level_t {
00971     TEK_SA_LOG_LEVEL_TRACE = 0x0,
00972     TEK_SA_LOG_LEVEL_DEBUG = 0x1,
00973     TEK_SA_LOG_LEVEL_INFO = 0x2,
00974     TEK_SA_LOG_LEVEL_WARNING = 0x3,
00975     TEK_SA_LOG_LEVEL_ERROR = 0x4,
00976     TEK_SA_LOG_LEVEL_CRITICAL = 0x5,
00977 };
00978
00985 /*****
00986  * Data client capabilities
00987  *****/
00988
00993 enum tek_sa_threading_model {
00998     TEK_SA_THREADING_MODEL_SAME_THREAD = 0x0,
00999
01004     TEK_SA_THREADING_MODEL_SEQUENTIAL = 0x1,
01005
01012     TEK_SA_THREADING_MODEL_PARALLEL = 0x2,
01013 };
01014
01023 struct tek_sa_data_client_capabilities {
01033     size_t number_of_inflight_calls;
01034
01039     enum tek_sa_threading_model threading_model;
01040 };
01041 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_capabilities, number_of_inflight_calls, 0, 0);
01042 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_capabilities, threading_model, 4, 8);
01043
01049 struct tek_sa_data_client {
01067     TEK_SA_RESULT (*register_features)(tek_sa_data_client_handle dc);
01068
01081     TEK_SA_RESULT (*connect)(tek_sa_data_client_handle dc);
01082
01088     void (*free)(tek_sa_data_client_handle dc);
01089
01182     TEK_SA_RESULT(*read_fields)(tek_sa_data_client_handle dc, uint64_t request_id,
01183     const tek_sa_field_handle items_to_read[], size_t number_of_items,
01184     bool do_not_block);
01185
01204     TEK_SA_RESULT(*write_fields)(tek_sa_data_client_handle dc, uint64_t request_id,
01205     const struct tek_sa_field_write_request items_to_write[],
01206     size_t number_of_items, bool do_not_block);
01207
01229     TEK_SA_RESULT(*block_read)(const tek_sa_data_client_handle dc, uint64_t request_id,
01230     const char* filepath, uint64_t offset, int64_t length, bool do_not_block,
01231     int64_t* filesize);
01232
01252     TEK_SA_RESULT(*block_write)(const tek_sa_data_client_handle dc, uint64_t request_id,
01253     const char* filepath, uint64_t offset, int64_t length, bool do_not_block);
01254
01272     TEK_SA_RESULT(*subscribe)(tek_sa_data_client_handle dc, const tek_sa_field_handle

```



```

    items_to_subscribe[],
01273     size_t number_of_items);
01274
01285 TEK_SA_RESULT(*unsubscribe)(tek_sa_data_client_handle dc,
01286     const tek_sa_field_handle items_to_unsubscribe[], size_t number_of_items);
01287
01311 TEK_SA_RESULT(*invoke)(const tek_sa_data_client_handle dc, const tek_sa_method_handle method,
01312     uint64_t request_id, const tek_sa_field_value parameters[],
01313     const size_t number_of_parameters);
01314
01330 void (*acknowledge_alarm)(tek_sa_data_client_handle dc,
01331     const tek_sa_alarm_handle alarm);
01332
01342 tek_sa_data_client_handle handle;
01343
01345 };
01346 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, register_features, 0, 0);
01347 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, connect, 4, 8);
01348 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, free, 8, 16);
01349 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, read_fields, 12, 24);
01350 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, write_fields, 16, 32);
01351 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, block_read, 20, 40);
01352 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, block_write, 24, 48);
01353 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, subscribe, 28, 56);
01354 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, unsubscribe, 32, 64);
01355 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, invoke, 36, 72);
01356 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, acknowledge_alarm, 40, 80);
01357 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, handle, 44, 88);
01358
01366 struct tek_sa_data_client_plugin {
01371     void* plugin_context;
01372
01389 TEK_SA_RESULT(*data_client_new)(void* plugin_context, const struct tek_sa_configuration* config,
01390     struct tek_sa_data_client* created_client,
01391     struct tek_sa_data_client_capabilities* capabilities);
01392
01396 void (*free_context)(void* plugin_context);
01397 };
01398 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, plugin_context, 0, 0);
01399 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, data_client_new, 4, 8);
01400 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, free_context, 8, 16);
01401
01414 struct tek_sa_transformation_engine {
01428     tek_sa_field_handle (*register_field)(tek_sa_data_client_handle dc,
01429         const char* name,
01430         enum tek_sa_field_attributes attributes,
01431         enum tek_sa_variant_type type);
01432
01446     tek_sa_method_handle (*register_method)(
01447         tek_sa_data_client_handle dc, const char* name,
01448         struct tek_sa_method_argument_description input_parameter[],
01449         size_t number_of_input_parameters,
01450         struct tek_sa_method_argument_description output_parameter[],
01451         size_t number_of_output_parameters);
01452
01465     tek_sa_event_handle (*register_event)(tek_sa_data_client_handle dc,
01466         const char* name);
01467
01481     tek_sa_alarm_handle (*register_alarm)(tek_sa_data_client_handle dc,
01482         const char* name,
01483         const int16_t severity,
01484         const tek_sa_field_handle source);
01485
01501     TEK_SA_RESULT(*register_enum_type)(tek_sa_data_client_handle dc,
01502         struct tek_sa_enum_definition const* type_definition,
01503         tek_sa_type_handle* result);
01504
01514     TEK_SA_RESULT(*register_struct_type)(tek_sa_data_client_handle dc,
01515         struct tek_sa_struct_definition const* type_definition,
01516         tek_sa_type_handle* result);
01517
01534     TEK_SA_RESULT(*post_event)(tek_sa_data_client_handle dc, struct tek_sa_dc_event const* event);
01535
01547     TEK_SA_RESULT(*set_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm);
01548
01557     TEK_SA_RESULT(*reset_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm);
01558
01581     void (*log)(tek_sa_data_client_handle source, enum tek_sa_log_level_t lvl,
01582         const char* format, va_list args);
01583
01597     tek_sa_event_handle (*get_global_event)(const char* name);
01598
01606     void (*update_capabilities)(
01607         tek_sa_data_client_handle dc,
01608         struct tek_sa_data_client_capabilities const* capabilities);
01609
01629     void (*read_progress)(tek_sa_data_client_handle dc, uint64_t request_id,

```

```

01630         uint64_t progress);
01631
01653 void (*read_result)(tek_sa_data_client_handle dc, uint64_t request_id,
01654                     TEK_SA_RESULT result,
01655                     const struct tek_sa_read_result results[],
01656                     size_t number_of_results);
01657
01665 void (*notify_change)(tek_sa_data_client_handle dc,
01666                       const struct tek_sa_read_result changes[],
01667                       size_t number_of_changes);
01668
01679 void (*write_result)(tek_sa_data_client_handle dc, uint64_t request_id,
01680                     TEK_SA_RESULT result,
01681                     const struct tek_sa_write_result results[],
01682                     size_t number_of_results);
01683
01695 void (*call_method_result)(tek_sa_data_client_handle dc, uint64_t request_id,
01696                           TEK_SA_RESULT result,
01697                           const struct tek_sa_field_value results[],
01698                           size_t number_of_results);
01699
01716 TEK_SA_RESULT(*block_read_data)(tek_sa_data_client_handle dc, uint64_t request_id, TEK_SA_RESULT
result,
01717                                unsigned char buffer[], size_t buffer_length);
01718
01733 TEK_SA_RESULT(*block_write_data)(tek_sa_data_client_handle dc, uint64_t request_id, unsigned char
buffer[],
01734                                   size_t buffer_length, size_t* bytes_written);
01735
01744 void (*block_write_result)(tek_sa_data_client_handle dc, uint64_t request_id,
01745                             TEK_SA_RESULT result);
01746
01748 };
01749 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_field, 0, 0);
01750 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_method, 4, 8);
01751 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_event, 8, 16);
01752 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_alarm, 12, 24);
01753 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_enum_type, 16, 32);
01754 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_struct_type, 20, 40);
01755 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, post_event, 24, 48);
01756 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, set_alarm, 28, 56);
01757 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, reset_alarm, 32, 64);
01758 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, log, 36, 72);
01759 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, get_global_event, 40, 80);
01760 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, update_capabilities, 44, 88);
01761 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, read_progress, 48, 96);
01762 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, read_result, 52, 104);
01763 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, notify_change, 56, 112);
01764 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, write_result, 60, 120);
01765 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, call_method_result, 64, 128);
01766 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_read_data, 68, 136);
01767 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_write_data, 72, 144);
01768 TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_write_result, 76, 152);
01769
01784 typedef TEK_SA_RESULT (*tek_sa_load_plugin_fn)(
01785     struct tek_sa_transformation_engine* api,
01786     const struct tek_sa_configuration* plugin_configuration,
01787     struct tek_sa_data_client_plugin* plugin);
01788
01789 #ifdef TEK_SA_DATA_CLIENT_IMPL
01790
01791 #ifdef _WIN32
01792 #define TEK_SA_API_EXPORT __declspec(dllexport) __stdcall
01793 #else
01794 #define TEK_SA_API_EXPORT __attribute__((__visibility__("default")))
01795 #endif
01796
01800 TEK_SA_RESULT TEK_SA_API_EXPORT
01801     load_plugin(struct tek_sa_transformation_engine* api,
01802                const struct tek_sa_configuration* plugin_configuration,
01803                struct tek_sa_data_client_plugin* plugin);
01804
01805 #endif
01806
01807 #ifdef __cplusplus
01808 }
01809 #endif
01810
01811 #undef TEK_SA_STRUCT_ALIGN_SELECT
01812 #undef TEK_SA_VERIFY_STRUCT_OFFSET
01813
01814 #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](#)