umati Transformation Engine - API documentation

(Release Candidate, 2021-10-01)

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.

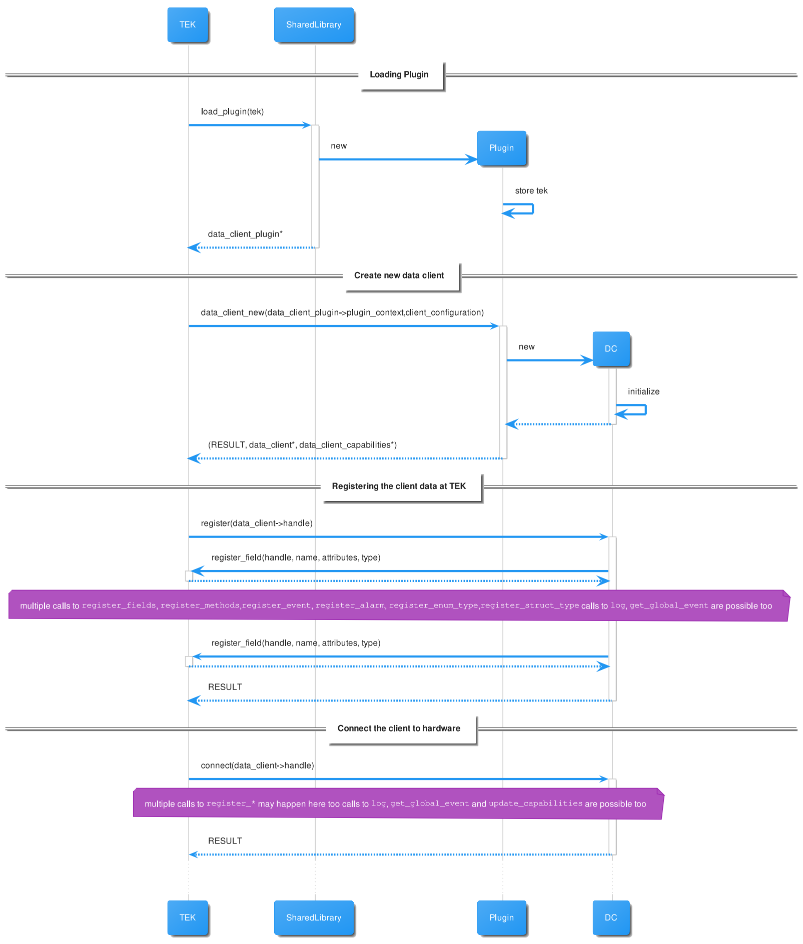
After loading the shared library the TEK calls the main initialization function with the fixed name `load_plugin` and a signature of tek_sa_load_plugin_fn . This function creates a new singleton instance of tek_sa_data_client_plugin and is expected to save the given TEK api struct.

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.


TEK
SharedLibrary
Loading Plugin
load_plugin(tek)
new
Plugin
store tek
data_client_plugin*
Create new data client
data_client_new(data_client_plugin->plugin_context,client_configuration)
new
DC
initialize
(RESULT, data_client*, data_client_capabilities*)
Registering the client data at TEK
register(data_client->handle)
register_field(handle, name, attributes, type)
multiple calls to register_fields, register_methods,register_event, register_alarm, register_enum_type,register_struct_type calls to log, get_global_event are possible too
register_field(handle, name, attributes, type)
RESULT
Connect the client to hardware
connect(data_client->handle)
multiple calls to register_* may happen here too calls to log, get_global_event and update_capabilities are possible too
RESULT
TEK
SharedLibrary
Plugin
DC

Chapter 3

Known issues

3.1 API definition issues

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

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

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

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

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

3.2 Documentation/Style issues

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

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

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

Chapter 4

Todo List

Page Known issues

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

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

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

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

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

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

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

Class tek_sa_complex_data_array

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

Class tek_sa_complex_data_matrix

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

Global tek_sa_data_client::read_fields)(tek_sa_data_client_handle dc, uint64_t request_id, const tek_sa↩_field_handle items_to_read[], size_t number_of_items, bool do_not_block)

[B, TEAM] define error values of read function

Global tek_sa_data_client::subscribe)(tek_sa_data_client_handle dc, const tek_sa_field_handle items_↩to_subscribe[], size_t number_of_items)

[D, TEAM] add sampling rate parameter

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

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

Class tek_sa_transformation_engine

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

Global tek_sa_transformation_engine::get_global_event)(const char ∗name)

[C, TEAM] define the predefined events

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

Global tek_sa_transformation_engine::read_progress)(tek_sa_data_client_handle dc, uint64_t request_id, uint64_t progress)

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

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

Global tek_sa_transformation_engine::set_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_↩handle alarm)

[C, TEAM] called by data_client after connect, regardless of "acknowledge" calls during previous connection?

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

8.1 Transformation Engine

Data Structures

- struct tek_sa_transformation_engine

    Interface ot 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_S
  = 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	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. **Remarks** This information may be dependent on the physical device and therefore available only after the connection was established.
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

api	The TEK api.
plugin_configuration	Additional configuration files, e.g. licensing information, for the plugin itself.
plugin	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	DLL is thread safe, multiple parallel calls are allowed. **Remarks** If the number of parallel tasks in the data client is reached, the API call may return ASYNC_RESULT_RETRY_LATER.

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 = 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, guids, 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 `data` 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 `data` 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 `status` 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

char const *	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	The Timestamp of the event. **Remarks** This should be the a value as close as possible to the actual occurrence of the event.
int16_t	severity	The severity level of the event. The severity is defined as in `https://reference.opcfoundation.org/v104/Core/docs/Part5/6.4.2/` which is cited here: 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.
tek_sa_event_handle	event_type	The event type handle as returned by the call to tek_sa_transformation_engine::register_event. **Attention** This field must not be TEK_SA_EVENT_HANDLE_INVALID
tek_sa_field_handle	source	The handle of the source of the event. 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.
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_d
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 wen 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

dc	data client handle features are registered for

This method is called from the TEK after the data client was created and before is 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 ist 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 retrive the values is larger then half the global timeout value a data client must call the vde_↩ sa_tek_ap::read_progress callback function.

Parameters

dc	The handle of the data client as returned from tek_sa_data_client_plugin::data_client_new.
request_id	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.
items_to_read	An array of field handles which describes the values the data client should read. See also function tek_sa_transformation_engine::register_field.
number_of_items	The number of handles in the parameter `items_to_read`.
do_not_block	A boolean flag that, when set to *true*, 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`

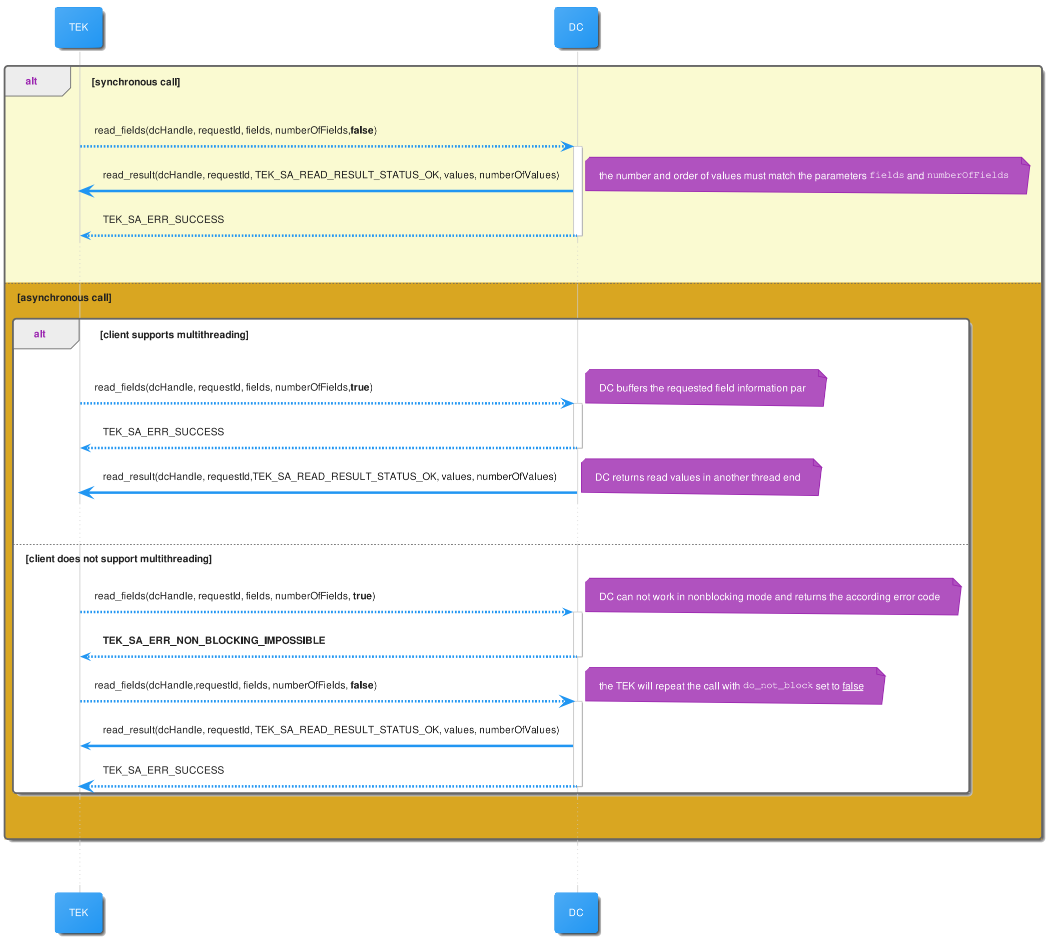

TEK
DC
alt
[synchronous call]
read_fields(dcHandle, requestId, fields, numberOfFields,**false**)
read_result(dcHandle, requestId, TEK_SA_READ_RESULT_STATUS_OK, values, numberOfValues)
the number and order of values must match the parameters `fields` and `numberOfFields`
TEK_SA_ERR_SUCCESS
[asynchronous call]
alt
[client supports multithreading]
read_fields(dcHandle, requestId, fields, numberOfFields,**true**)
DC buffers the requested field information par
TEK_SA_ERR_SUCCESS
read_result(dcHandle, requestId,TEK_SA_READ_RESULT_STATUS_OK, values, numberOfValues)
DC returns read values in another thread end
[client does not support multithreading]
read_fields(dcHandle, requestId, fields, numberOfFields, **true**)
DC can not work in nonblocking mode and returns the according error code
TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE
read_fields(dcHandle,requestId, fields, numberOfFields, **false**)
the TEK will repeat the call with `do_not_block` set to false
read_result(dcHandle, requestId, TEK_SA_READ_RESULT_STATUS_OK, values, numberOfValues)
TEK_SA_ERR_SUCCESS
TEK
DC

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

dc	The handle of the data client as returned from tek_sa_data_client_plugin::data_client_new.
request_id	A unique request identifier which is created by the TEK and must be passed to call to tek_sa_transformation_engine::write_result.
items_to_write	An array of field handles and their values which describes the values the data client should write.
number_of_items	The number of handles in the parameter `items_to_write`.
do_not_block	A boolean flag that, when set to *true*, 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

dc	The data client handle
request_id	The request id for the TEK API callbacks
filepath	The file or address of the block to be read. The format is data client specific. The pointer must be in utf-8.
offset	The offset in the data
length	A specific length, or -1 for the whole data
do_not_block	See Usage in `read_fields`
filesize	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

dc	The handle of the data client as returned from tek_sa_data_client_plugin::data_client_new.
request_id	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.
offset	The offset in the data
length	A specific length, or -1 for the whole data
do_not_block	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

dc	The handle of the data client as returned from tek_sa_data_client_plugin::data_client_new.
items_to_subscribe	The fields for which change events will be received.
number_of_items	The number of elements in the `items_to_subscribe` 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

dc	The handle of the data client as returned from tek_sa_data_client_plugin::data_client_new.
items_to_unsubscribe	The fields for which no more change events will be received.
number_of_items	The number of elements in the `items_to_unsubscribe` 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

dc	The handle of the data client as returned from tek_sa_data_client_plugin::data_client_new.
method	The method handle which is returned from the tek_sa_data_transformation_engine::register_method method.
request_id	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.
parameters	The parameters of the method. Number and type must match the method registration.
number_of_parameters	The number of parameters in the `parameters` 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

dc	The handle of the data client as returned from tek_sa_data_client_plugin::data_client_new.
alarm	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_capabilitie
*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

plugin_context	
config	
created_client	
capabilities	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 ot 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 ot 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

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

dc	The data client that registers at the TEK.
name	The name of the method.
tek_sa_method_argument_description	The description of the method input arguments.
number_of_input_parameters	The number of input parameters.
tek_sa_method_argument_description	The description of the method output arguments.
number_of_output_parameters	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

dc	The data client that registers at the TEK.
name	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 int16_t severity, const tek_sa_field_handle source)
```

Registers an alarm at the TEK.

Parameters

dc	The data client that registers at the TEK.
name	The name of the new alarm, must be unique within all alarms registered for this data client.
severity	The alarm severity level.
source	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

dc	The data client that registers at the TEK.
tek_sa_enum_definition	The definition of the enumeration.
result	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

dc	The data client that registers at the TEK.
tek_sa_struct_definition	The definition of the struct.
result	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

dc	Handle of the data client which sends the event.
event	A event structure. See `dc_event`.

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

dc	Handle of the data client that sets the alarm.
alarm	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

dc	Handle of the data client that clears/resets the alarm.
alarm	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

data_client_handle	The data client that logs a message.
lvl	The logging level.
format	The message format string. Format must be compatible to `printf`.
args	A va_list 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

name	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

dc	Handle of the data client that informs about the change of its capabilities.
tek_sa_data_client_capabilities	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

dc	Handle of the data client that is the source of the call
request↩ _id	id of request to data client which triggered the call back
progress	?? (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

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

dc	Handle of the data client that is the source of the change
changes	changed field values
number_of_changes	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

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

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

dc	The data client handle.
request_id	The request id of the block transfer.
result	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.
buffer	The current chunk of the file. The TEK must copy the data into it's own process.
buffer_length	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

dc	The data client handle.
request_id	The request id of the block transfer.
buffer	The buffer to write the chunk of the file. The TEK must copy the data into the buffer provided by the data client.
buffer_length	The length of the buffer in the data client.
bytes_written	The number of bytes written in the buffer by the TEK.
result	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

dc	The data client handle.
request←_id	The request id of the block transfer.
result	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 ot 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 = 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)))};
```

```
#endif

#ifdef __cplusplus
extern "C" {
#endif

typedef void* tek_sa_data_client_handle;

/****************************************************************************************************
 * Configuration structures
 ****************************************************************************************************/

struct tek_sa_additional_file {
  char* name;

  char* content;
};

TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_additional_file, name, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_additional_file, content, 4, 8);

struct tek_sa_configuration {
  char* config;

  struct tek_sa_additional_file* additional_files;

  size_t additional_files_count;
};

TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, config, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, additional_files, 4, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_configuration, additional_files_count, 8, 16);

/****************************************************************************************************
 * Built-in type definitions and variant
 ****************************************************************************************************/

typedef int64_t tek_sa_type_handle;

typedef int64_t tek_sa_type_handle_or_type_enum;

struct tek_sa_guid {
  uint32_t data1;

  uint16_t data2;

  uint16_t data3;

  uint8_t data4[8];
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data1, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data2, 4, 4);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data3, 6, 6);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_guid, data4, 8, 8);

struct tek_sa_byte_string {
  int32_t length;

  unsigned char* data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_byte_string, length, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_byte_string, data, 4, 8);

struct tek_sa_string {
  int32_t length;

  unsigned char* data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_string, length, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_string, data, 4, 8);

typedef int64_t tek_sa_datetime;

struct tek_sa_complex_data {
  tek_sa_type_handle type;

  uint32_t data_length;

  unsigned char* data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, type, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, data_length, 8, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data, data, 12, 16);

struct tek_sa_complex_data_array_item {
  uint32_t data_length;

  unsigned char* data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array_item, data_length, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array_item, data, 4, 8);

struct tek_sa_complex_data_array {
  tek_sa_type_handle type;

  size_t number_of_items;

  struct tek_sa_complex_data_array_item* data;
};

TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, type, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, number_of_items, 8, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_array, data, 12, 16);

struct tek_sa_complex_data_matrix {
  tek_sa_type_handle type;

  int32_t dimension_length;

  int32_t* dimensions;

  struct tek_sa_complex_data_array_item* data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, type, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, dimension_length, 8, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, dimensions, 12, 16);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_complex_data_matrix, data, 16, 24);

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
};

struct tek_sa_variant_array {
  int32_t length;

  union {
    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;
  } data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_array, length, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_array, data, 4, 8);

struct tek_sa_variant_matrix {
  int32_t dimension_length;

  int32_t* dimensions;

  struct tek_sa_variant_array data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_matrix, dimension_length, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant_matrix, dimensions, 4, 8);

struct tek_sa_variant {
  uint8_t type;

  union {
    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;
  } data;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant, type, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_variant, data, 8, 8);

typedef struct tek_sa_variant tek_sa_field_value;

/**************************************************************************************************
 * Type definitions from data client to TEK
 *************************************************************************************************/

struct tek_sa_struct_field_type_definition {
  char* name;

  tek_sa_type_handle_or_type_enum type;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_field_type_definition, name, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_field_type_definition, type, 8, 8);

struct tek_sa_struct_definition {
  char* name;

  struct tek_sa_struct_field_type_definition* items;

  size_t item_count;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, name, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, items, 4, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_struct_definition, item_count, 8, 16);

struct tek_sa_enum_item_definition {
  char* name;

  int32_t value;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_item_definition, name, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_item_definition, value, 4, 8);

struct tek_sa_enum_definition {
  char* name;

  struct tek_sa_enum_item_definition* items;

  size_t item_count;
};
```

```
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, name, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, items, 4, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_enum_definition, item_count, 8, 16);

struct tek_sa_method_argument_description {
  char const* name;

  enum tek_sa_variant_type type;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_method_argument_description, name, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_method_argument_description, type, 4, 8);

/*************************************************************************************************
 * Handles and structures for data exchange
 *************************************************************************************************/

enum tek_sa_field_attributes {
  TEK_SA_FIELD_ATTRIBUTES_WRITABLE = 0x1,

  TEK_SA_FIELD_ATTRIBUTES_READABLE = 0x2,

  TEK_SA_FIELD_ATTRIBUTES_SUBSCRIBABLE = 0x4,
};

typedef uint32_t tek_sa_field_handle;

typedef uint32_t tek_sa_event_handle;

typedef uint32_t tek_sa_alarm_handle;

typedef uint32_t tek_sa_method_handle;

#define TEK_SA_FIELD_HANDLE_INVALID 0

#define TEK_SA_EVENT_HANDLE_INVALID 0

#define TEK_SA_ALARM_HANDLE_INVALID 0

#define TEK_SA_METHOD_HANDLE_INVALID 0

typedef int TEK_SA_RESULT;

#define TEK_SA_ERR_SUCCESS 0

#define TEK_SA_ERR_NON_BLOCKING_IMPOSSIBLE 10

#define TEK_SA_ERR_OUT_OF_MEMORY 11

#define TEK_SA_ERR_INVALID_PARAMETER 12

#define TEK_SA_ERR_RETRY_LATER 0xffffffff

#define TEK_SA_READ_RESULT_STATUS_OK 0

#define TEK_SA_READ_RESULT_STATUS_NOK 1

#define TEK_SA_READ_RESULT_STATUS_TIMEOUT 2

#define TEK_SA_READ_RESULT_STATUS_INVALID_HANDLE 3

#define TEK_SA_BLOCK_TRANSFER_END_OF_FILE 26

#define TEK_SA_BLOCK_TRANSFER_ABORT 24
/*************************************************************************************************
 * Request and response structures
 *************************************************************************************************/

struct tek_sa_field_write_request {
  tek_sa_field_handle handle;

  tek_sa_field_value value;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_field_write_request, handle, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_field_write_request, value, 8, 8);

struct tek_sa_write_result {
  TEK_SA_RESULT status;

  tek_sa_field_handle handle;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_write_result, status, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_write_result, handle, 4, 4);

struct tek_sa_read_result {
  TEK_SA_RESULT status;

  tek_sa_field_handle handle;
  tek_sa_field_value value;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, status, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, handle, 4, 4);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_read_result, value, 8, 8);

struct tek_sa_event_parameter {
  char const* name;

  tek_sa_field_value value;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_event_parameter, name, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_event_parameter, value, 8, 8);

struct tek_sa_dc_event {
  tek_sa_datetime timestamp;

  int16_t severity;

  tek_sa_event_handle event_type;

  tek_sa_field_handle source;

  size_t number_of_parameters;

  struct tek_sa_event_parameter* parameters;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, timestamp, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, severity, 8, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, event_type, 12, 12);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, source, 16, 16);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, number_of_parameters, 20, 24);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_dc_event, parameters, 24, 32);

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,
};

/***************************************************************************************************
 * Data client capabilities
 **************************************************************************************************/

enum tek_sa_threading_model {
  TEK_SA_THREADING_MODEL_SAME_THREAD = 0x0,

  TEK_SA_THREADING_MODEL_SEQUENTIAL = 0x1,

  TEK_SA_THREADING_MODEL_PARALLEL = 0x2,
};

struct tek_sa_data_client_capabilities {
  size_t number_of_inflight_calls;

  enum tek_sa_threading_model threading_model;
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_capabilities, number_of_inflight_calls, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_capabilities, threading_model, 4, 8);

struct tek_sa_data_client {
  TEK_SA_RESULT (*register_features)(tek_sa_data_client_handle dc);

  TEK_SA_RESULT (*connect)(tek_sa_data_client_handle dc);

  void (*free)(tek_sa_data_client_handle dc);

  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);

  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);

  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);

  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);

  TEK_SA_RESULT(*subscribe)(tek_sa_data_client_handle dc, const tek_sa_field_handle
 items_to_subscribe[],
   size_t number_of_items);

  TEK_SA_RESULT(*unsubscribe)(tek_sa_data_client_handle dc,
   const tek_sa_field_handle items_to_unsubscribe[], size_t number_of_items);

  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);

  void (*acknowledge_alarm)(tek_sa_data_client_handle dc,
                            const tek_sa_alarm_handle alarm);

  tek_sa_data_client_handle handle;

};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, register_features, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, connect, 4, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, free, 8, 16);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, read_fields, 12, 24);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, write_fields, 16, 32);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, block_read, 20, 40);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, block_write, 24, 48);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, subscribe, 28, 56);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, unsubscribe, 32, 64);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, invoke, 36, 72);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, acknowledge_alarm, 40, 80);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client, handle, 44, 88);

struct tek_sa_data_client_plugin {
  void* plugin_context;

  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);

  void (*free_context)(void* plugin_context);
};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, plugin_context, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, data_client_new, 4, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_data_client_plugin, free_context, 8, 16);

struct tek_sa_transformation_engine {
  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);

  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);

  tek_sa_event_handle (*register_event)(tek_sa_data_client_handle dc,
                                        const char* name);

  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);

  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);

  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);

  TEK_SA_RESULT(*post_event)(tek_sa_data_client_handle dc, struct tek_sa_dc_event const* event);

  TEK_SA_RESULT(*set_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm);

  TEK_SA_RESULT(*reset_alarm)(tek_sa_data_client_handle dc, const tek_sa_alarm_handle alarm);

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

  tek_sa_event_handle (*get_global_event)(const char* name);

  void (*update_capabilities)(
      tek_sa_data_client_handle dc,
      struct tek_sa_data_client_capabilities const* capabilities);

  void (*read_progress)(tek_sa_data_client_handle dc, uint64_t request_id,
                        uint64_t progress);

  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);

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

  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);

  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);

  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);

  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);

  void (*block_write_result)(tek_sa_data_client_handle dc, uint64_t request_id,
                             TEK_SA_RESULT result);

};
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_field, 0, 0);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_method, 4, 8);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_event, 8, 16);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_alarm, 12, 24);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_enum_type, 16, 32);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, register_struct_type, 20, 40);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, post_event, 24, 48);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, set_alarm, 28, 56);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, reset_alarm, 32, 64);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, log, 36, 72);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, get_global_event, 40, 80);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, update_capabilities, 44, 88);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, read_progress, 48, 96);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, read_result, 52, 104);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, notify_change, 56, 112);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, write_result, 60, 120);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, call_method_result, 64, 128);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_read_data, 68, 136);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_write_data, 72, 144);
TEK_SA_VERIFY_STRUCT_OFFSET(tek_sa_transformation_engine, block_write_result, 76, 152);

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);

#ifdef TEK_SA_DATA_CLIENT_IMPL

#ifdef _WIN32
#define TEK_SA_API_EXPORT __declspec(dllexport) __stdcall
#else
#define TEK_SA_API_EXPORT __attribute__((__visibility__("default")))
#endif

TEK_SA_RESULT TEK_SA_API_EXPORT
    load_plugin(struct tek_sa_transformation_engine* api,
            const struct tek_sa_configuration* plugin_configuration,
            struct tek_sa_data_client_plugin* plugin);

#endif

#ifdef __cplusplus
}
#endif

#undef TEK_SA_STRUCT_ALIGN_SELECT
#undef TEK_SA_VERIFY_STRUCT_OFFSET

#endif /* TEK_SOUTH_API_H */
```

Index