SINTEF

SINTEF Energy Research

Address: NO-7465 Trondheim, NORWAY
Reception: Sem Sælands vei 11
Telephone: +47 73 59 72 00
Telefax: +47 73 59 72 50

www.energy.sintef.no

Enterprise No.:
NO 939 350 675 MVA

TECHNICAL REPORT

SUBJECT/TASK (title)

ELCBAS/SEA Programmers Guide

CONTRIBUTOR(S)

Tormod Lund, Nils Eggen, Birger Stene

CLIENTS(S)

Joint project: ABB Kraft AS, Siemens AS, Sintef Energy Research AS, Statnett SF

TR NO.	DATE	CLIENT'S REF.	PROJECT NO.
TR A5833	2003-12-22		12X246

ELECTRONIC FILE CODE	RESPONSIBLE (NAME, SIGN.)	CLASSIFICATION
030620bs12583	Ove Grande	Unrestricted

ISBN N0.	REPORT TYPE	RESEARCH DIRECTOR (NAME, SIGN)	COPIES	PAGES
82-594-2506-8		Petter Støa	10	77

DIVISION	LOCATION	LOCAL FAX
Energy Systems	Sem Sælandsv. 11	+47 73 59 72 50

RESULT (summary)

This document describes programming data sources and data sinks for the Elcbas/SEA system using the Simplified Elcom API (SEAPI).

Please see SINTEF's homepage at: http://www.sintef.no/ELCOM-90. From here you can download the latest version of all relevant documents as pdf-files for free.

KEYWORDS

SELECTED BY AUTHOR(S)		
	Data communication	Communication protocols
	Control Centres	ELCOM-90

Document Identity: TR A5833 Revision: 05

Technical Reference:

Approved by:

Attested by:

Date	Revision	Synopsis
2003-06-10	01	New Document
2003-09-24	02	Added/reviewed reference information for functions.
2003-12-22	03	First Official Release
2004-03-02	04	Updated after comments
2011-01-10	05	Updated for Elcbas V6.5

Table of Contents

Table of Figures

Preface

Purpose

The purpose of this document is to describe programming data sources and data sinks for the Elcbas/SEA system using the Simplified Elcom API (SEAPI).

Intended Audience

This is a technical document. Some familiarity with Elcom concepts and with the C programming language is assumed

Structure of the Document

The first part of this document is a brief tutorial description of programming the SEAPI, whereas the last part contains reference information.

Associated Documents

1. TR A5835 – Elcbas/SEA for Windows Administrators Guide
2. TR A5834 – Elcbas/SEA Configuration Guide
3. TR A3825 – Elcom User Element Conventions

Consult [3] for a more extensive list of Elcom documentation.

Acronyms

SEA	Simplified Elcom API
API	Application Programming Interface
SEAIN	The initiator in the Elcbas/SEA system
SEARS	The responder in the Elcbas/SEA system

Trademarks

None

1. Introduction

The Elcbas Simplified Elcom API, SEAPI, allows an application to send and receive Elcom data, as well as commands and setpoints. The handling of the Elcom protocol details is done by the Elcbas runtime system, and the interface to the application is through lists of name/value pairs. For a particular installation, there may be one or more SEAPI applications, or clients. The routing of data is determined by the configuration of the Elcbas runtime system.

The figure below shows an example system with a single SEAPI client application, Client1. It also illustrates the different types of data flow supported by the API.

An SEAPI client can fulfill one or more of the following roles:

- Data Source – maintains data which the responder (sears) sends to one or more Elcom partners
- Data Sink – receives data from the initiator, which has received these data from one or more Elcom partners
- Command Source – sends commands or setpoints to one or more remote partners through the initiator
- Command Sink – receives commands or setpoints from one or more Elcom partners through the responder
- Management Application – controls the operation of the Elcom runtime system through the API. For this version, the management functionality is considered internal, and is not intended for use by third party client applications.

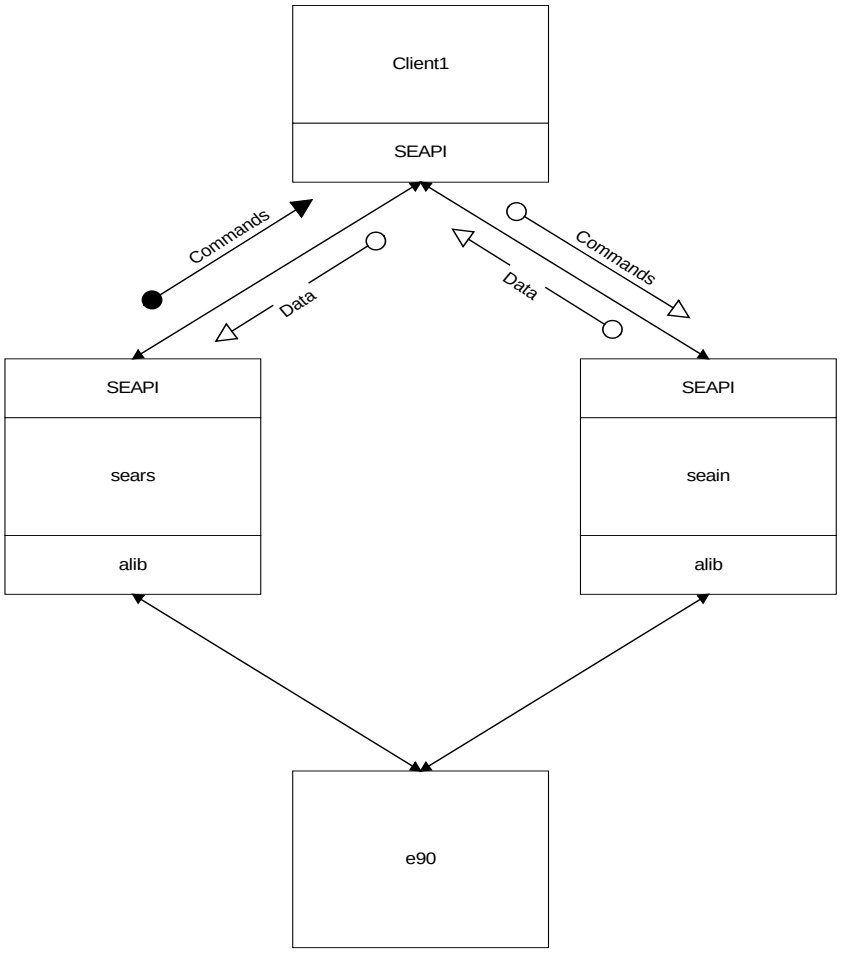

Figure 1 Sample Elcbas System

The SEAPI library is designed as an API for the C programming language, and should also be fully usable from C++. For the Windows platform a Visual Basic Wrapper (for VB6) is available upon request.

2. Using the SEAPI

2.1 General Program Flow

The SEAPI is generally designed as a request-response style API, in which the server applications sends requests to the client applications, which may or may not need to provide a response. Responses, when needed, must be provided in a timely fashion. Within this model, the API should allow considerable flexibility in usage. A typical, simple, client may look like, in a simplified form:

```
//
// Sample showing flow only
//

#include <stdio.h>
#include <stdlib.h>
#include <seapub.h>

static void die (char *msg)
{
  fprintf (stderr, "%s\n", msg);
  exit (2);
}

int main (int argc, char **argv)
{
  SEAStatus status;
  SEAHandle data = SEA_NULL_HANDLE;

  if ((status = SEAOpen ("client1", SEA_FN_AUTO, 0)) != SEA_OK)
    die ("Failed to open the API");

  for (;;) {
    SEARequest rqId;

    rqId = SEAGetRequest (SEA_RQ_ALL, SEA_INFINITE, &data);

    switch (rqId) {
    case SEA_RQ_DATA:
      fprintf (stderr, "Data received\n");
      break;

    case SEA_RQ_SHUTDOWN:
      die ("API returned SHUTDOWN");
      break;

    default:
      fprintf (stderr, "Unexpected request code %x\n", rqId);
      break;
    }
  }
}
```

This illustrates:

1. An application must always call SEAOpen prior to any other API call. The first parameter is the client application name, of which the first 8

characters are significant, and must be unique in a running system (ie. only one instance of an application may be active at any time).

2. The application receives data or requests for data through the SEAGetRequest call. The second parameter is a timeout in seconds, which may be 0 for use in a polling mode, or the constant SEA_INFINITE shown here, which means 'wait until something is available'.
3. The API maintains memory internally using handles. Functions that may cause the allocation or deallocation of memory require that the address of the handle is passed, other functions use just the handle value. If calling a function like SEAGetRequest, the previous contents of the handle will be automatically released by the API. This may also be done using the SEAFree function. The handle must initially have the value SEA_NULL_HANDLE.
4. In case of certain errors, the SEAGetRequest function will return SEA_RQ_SHUTDOWN. The caller must then either exit, or may attempt to call SEAClose and then SEAOpen again.
5. Since the routing of data is set up in the configuration files for the initiator and responder, the client application should always be prepared to deal with unsupported request codes in a graceful fashion (ie. not crash).

2.2 Data Identification

Objects are identified in the API using names as configured with the Elcbas Configuration tool. Earlier, names have been restricted to a maximum length of 40 characters (the constant SEA_NAMELEN). With the new Ex functions, this restriction is removed, and names can, in principle, be of any length. Names may consist of any printable character.

Note that, whereas the old SEAObject struct contains the name as an embedded character array, so that a copy of the structure also copies the name, the new SEAObjectEx structure only contains a pointer to the name, so that this must be copied separately, if needed.

Objects have a data type corresponding to the Elcom type, with the following main types:

- 32-bit (single precision) floating point, and corresponding setpoints
- 16-bit signed integer, and corresponding setpoints
- 2-bit status (on, off, between)
- commands (on, off)

2.3 Programming a Data Source Application

A data source application is responsible for delivering data to the responder upon request so that the responder can deliver these data to its Elcom partners.

There are three different modes of operation for a data source application:

- Unsolicited or event-driven data
- Periodic polled data
- Requested data

2.3.1 Unsolicited Data Transfer

When delivering unsolicited data, an application needs to service subscription requests from the responder, in the form of request codes:

- SEA_RQ_START_UNSOLICITED – start unsolicited data transfer for the supplied list of objects. The application should fill inn the current value and quality for each of the objects and return the list with a response code of SEA_RS_UNSOLICITED_STARTED.
- SEA_RQ_STOP_UNSOLICITED – stop unsolicited data transfer for the supplied list of objects. No response is required.

If the API was opened in V1 mode (a function code of SEA_FN_UNSOLICITED was supplied), the api will generate a SEA_RQ_STOP_UNSOLICITED, with a list id of SEA_ID_ALL in the header and an empty list, whenever the responder stops.

If using SEA_FN_RSCLIENT in SEAOpen, this will not happen, use the option flag SEA_OP_MONITOR instead, and handle SEA_RQ_PARTNER_DOWN messages if you need to handle this case.

Once subscription is established, data is sent by:

- Creating an object list and adding names and values to it. Up to 255 objects may be added to one list.
- Send the list to the responder with the request code SEA_RS_UNSOLICITED.
    - For V1 compatibility, using SEASendResponse with such a list will cause the list to be sent to the responder.
    - It is recommended that you use the SEASendRequestTo for new code, to make the function clearer. The constant SEA_SRV_RS defines the responder api name for use with this function.

Note that updates may be sent from any client/thread, not just the one handling START_UNSOLICITED. To enable this, consider the following:

- For V1 clients, this is done by using a blank function mask in SEAOpen for the client sending the changes.
- For newer clients, this is done by using the same object directory for the clients. Ensure that the client handling start/stop is the one configured for unsolicited data for the partner in question (or system wide).

2.3.2 Periodic Data Transfer

For periodic data transfer the responder will poll the configured client at the configured interval, using a request code of SEA_RQ_PERIODIC_DATA. The application should fill the supplied list with data, and return it with a response code of SEA_RS_DATA.

To enable caching of data access, the application may read the header to get some hints:

- The field identifier is a generated number which will guarantee (within reason) that the list of objects is the same and in the same order.
- The dataType indicates the data type of all objects (which will be the same).
- The source will equal to SEA_FU_PERIODIC.
- The period will be the configured period, in seconds (this is for information, it is the responder which will run the timer).

As there is currently no 'stop' request available for periodic, an application caching access info should have some form of 'garbage collection' on this, throwing away lists that have not been seen for some time (the period setting can be used for this, but more than one period should be used, as the period does not include any processing delays).

2.3.3 Requested Data Transfer

For requested data transfer the responder will send a request for each request received from the partner. In this case the application should always read the object list header, considering the following fields:

- timeSpec – if this is NULL, this is a request for 'latest data', otherwise it points to a SEATimeSpec structure detailing the data wanted. This has the fields:
    - T0 – the time of the first data set (incarnation)
    - dt – the interval between data sets
    - dtUnit – the time unit for dt (enum SEATimeUnit)
    - periods – the number of data sets
    - curPeriod – the current period
- source – may be one of SEA_FU_SCHEDULED for plan values or SEA_FU_ARCHIVED for archived/current values. The exact interpretation of this is up to the client application.
- dataType – is the type for the objects in the list
- identifier – will be different for each invocation

The application must respond to this request by filling in values and sending the list back with the response code SEA_RS_DATA.

If a time series is required, this must be repeated the number of times given in periods. The same list must be used for all responses. The following techniques may be helpful:

- All values in the list must be time stamped according to the requested time series. The api has the function SEAAddToTime which may be used to calculate the time for the next data set.
- The current period in the header must be increased prior to each response (it is initially 0). This may be done using the api call SEANextPeriod.
- If you prefer to navigate the object list using SEAObjlistNext, the current pointer of the list may be reset using SEAObjlistReset.

The application may reject a requested data transfer by setting a result code using SEASetResult and responding to the request with SEA_RQ_ERROR_RESPONSE. This should be done only once pr. request.

The following result codes are allowed:

- SEA_RES_NO_DATA – The requested list of objects or time specification matched no data in the application. Note that if some data may be returned, they should, with the remaining incarnations returned as default values with bad quality.
- SEA_RES_ILLEGAL_INTERVAL – The specified sample interval is not available in the application.

2.4 Programming a Data Sink Application

A data sink application receives data from the initiator as it receives it from remote partners. All data are received with the request code SEA_RQ_DATA, and simple clients may just process each item in the list individually. To enable caching of database access, sophisticated clients may read the header to get hints as follows:

- identifier – if this is 0, the object list should be considered transient. Also, the objects in the list may be of different types, so the dataType should be read for each object (dataType in the header will also be 0). If the identifier is not 0, it signifies a particular set of data, which should not change as long as the identifier is the same.
- dataType – if not 0 indicates the type of the objects in the list and that these are all the same.
- source – if not 0, this is one of SEA_FU_PERIODIC, SEA_FU_SCHEDULED or SEA_FU_ARCHIVED according to the transfer method and suffix used.

- timeSpec – if not NULL will contain the time information for the current data incarnation (see above for a description of the timeSpec fields).

For periodic and requested data transfer, each SEA_RQ_DATA contains exactly one set of data (one incarnation).

There is no 'stop' available to indicate when a specific set of data is no longer relevant, so if some caching is used, the application should garbage collect the cache as needed.

2.5 Programming a Command Source Application

A command source application sends commands and setpoints to remote applications via the initiator.

A single command or multiple setpoints may be sent in one request. If multiple setpoints are sent, these must be defined at consecutive indices in a setpoint group, and supplied in the object list in group order.

- The application should create an object list, fill in the relevant command/setpoint data, and send to the initiator with a request code of SEA_RQ_SETPOINT or SEA_RQ_COMMAND.
- Note that the objects must have data types of SEA_DT_COMMAND, SEA_DT_SPT_FLOAT or SEA_DT_SPT_INT, rather than the data types used for receiving data.
- The application should wait for a response from the remote side, which will come as an object list with a response code of SEA_RQ_COMMAND_RESPONSE or SEA_RQ_SETPOINT_RESPONSE.
- The result of the operation will be in the quality code of the object(s) in the returned list.
- Note that the initiator will return with quality SEA_Q_ILLOBJ if the supplied object(s) are not configured properly as commands/setpoints of the correct type, or if multiple setpoints are given which are not consecutive in a group. This code may also be returned from the remote side, if this is an Elcbas/SEA system.

Note that the command response only indicates that this is a plausible command or setpoint in the remote system, any data changes resulting from this must be configured as normal data objects in order to be received.

2.6 Programming a Command Sink Application

A command sink application receives commands and setpoints from Elcom partners via the responder.

The application will receive a single command or one or more setpoints of the same type with the request code SEA_RQ_COMMAND and SEA_RQ_SETPOINT respectively.

The application should inspect the command/setpoint data, and respond by updating the quality code(s) and returning the list with a response code of either SEA_RS_COMMAND_RESPONSE or SEA_RS_SETPOINT_RESPONSE.

The result should indicate whether the command/setpoint(s) are acceptable in the application or not. Any resulting changes must be updated separately as data (assuming that the remote partner has configured this).

Special quality codes are defined for command responses, see the reference section.

2.7 Programming the Management Interface

The Management Interface is currently considered proprietary. It may be documented in a future version.

2.8 Compiling, Linking and Running your Application

2.8.1 Windows Platform

For the windows platform, follow these steps:

- Add the directory <install>\include to your compiler include file search path, where <install> is wherever the Elcbas/SEA software was installed (typically C:\Program Files\Elcbas).
- Add the file seapi.lib to your linker inputs, and add the directory <install>\lib to the linkers search path.
- Add the directory <install>\bin to the executable search path for the runtime environment of your target application. You should avoid copying the seapi.dll to your own directory, as this may create incompatibilities with future upgrades (a new dll will be compatible, but the protocol between applications may change, as this is hidden by the dll).

If your compiler is incompatible with the Microsoft Visual C++ name mangling for stdcall routines, you may need to access the functions by name, using LoadLibrary and GetProcAddress or similar techniques. A list of function names and ordinal numbers is available upon request.

2.8.2 Unix/Linux Platforms

Please consult the release notes supplied with your kit.

3. Reference

3.1 Functions

SEAAddInput

Add user-supplied event sources to the wait mechanism in SEAGetRequest

Synopsis

```
SEAStatus SEAAddInput (SEAInput input);
```

Arguments

input
This is the input source descriptor (see below for platform specific descriptions).

Return Values

SEA_OK
Normal, successful return.

SEA_ILLINP
The input source is invalid, or the maximum number of inputs is reached.

Description

This function may be used to add event sources such as file descriptors to the event handling in SEAGetRequest. Typically this function is used to synchronize with the source of data changes when supporting unsolicited data transfer

- For Unix systems this is file descriptors e.g. for ipc channels, that are usable in a select (2) call.
- For Win32 supports any object that may be waited on in WaitForMultipleObjects.

Example

Win32:

```
HANDLE event;

event = CreateEvent(NULL, FALSE, FALSE, “DATA_CHANGED”);
if (SEAAddInput (event) != SEA_OK) {
        fprintf (stderr, “Unable to add input\n”);
}
```

Unix:

```
int fd;
fd = open (“/tmp/pipe”, O_RDWR, 0);
if (SEAAddInput (fd) != SEA_OK) {
```

```
            fprintf (stderr, “Unable to add input\n”);
}
```

Notes

None

SEAAddToTime

Add an interval to a time in SEAPI format.

Synopsis

```
SEAStatus SEAAddToTime (SEATime *time,
                        int dt,
                        SEATimeUnit unit);
```

Arguments

time
[in, out] The time to add to.

dt
[in] The interval to add.

unit
[in] The unit of the interval, members of the enumerator SEATimeUnit.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
The time pointer is invalid or the unit is outside the enumerator range.

Description

This function will add an interval to the supplied time. If the unit is SEA_TU_MONTH or SEA_TU_YEAR, the calculation will only affect the month and/or year fields. In other cases, the interval will be added as actual elapsed time.

Example

See the sample t_rs_data.

Notes

This function is new in SEAPI version 3.

SEAAlibTimeToSEATime

Convert time from Elcom alib format (integer array) to SEAPI format.

Synopsis

```
SEAStatus SEAAlibTimeToSEATime (int altime[7],
                                SEATime *seatime);
```

Arguments

altime
[in] The time in Elcom alib format, ie. an integer array, with year as year – 1900.

seatime
[out] The same time in SEAPI format.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
The seatime pointer is invalid.

Description

This function will convert a time in the format used by the Elcom-90 API (alib) to the format used by the SEAPI. This is a straight copy of data from the integer array, with the exception of the year field, which is based from 1900 in the Elcom format.

Example

None available.

Notes

This function is new in SEAPI version 3.

SEAClose

Terminate the SEAPI and free associated resources. After this function is called no other API functions should be used,

Synopsis

```
void SEAClose ();
```

Arguments

None.

Return Values

None.

Description

The SEAClose function is used to properly terminate an applications association with the API. It will result in all communication handled by this application to stop, and all api internal resources to be deallocated.

The application may call SEAOpen again to reestablish communication with the api.

Example

```
SEAClose (); /* sic! */
```

Notes

The function may always safely be called. It is harmless even if SEAOpen wasn't called earlier.

SEACominfoGet

Get connection state information

Synopsis

```
SEAStatus SEACominfoGet (SEAHandle reqInfo,
                         SEACominfo **infoptr);
```

Arguments

reqInfo
The handle we used in the SEAGetRequest call.

infoptr
The connection information may be read from the structure pointed to by this pointer. Note: This memory is maintained by the API, and will be invalidated after the next use of the related handle.

Return Values

SEA_OK
Normal, successful return.

SEA_ILLHANDLE
The handle did not refer to a connection info event.

Description

The SEACominfoGet is used to read out the information after a SEA_RQ_COMMS_EVENT request .

Example

```
SEACominfo *ciptr;
if (SEACominfoGet (handle, &ciptr) != SEA_OK) {
  fprintf (stderr,
           "Failed to read connection information\n");
}
```

Notes

This function is deprecated and should not be used.

SEAConnect

Connect explicitly to an api partner.

Synopsis

```
SEAStatus sts = SEAConnect (const char *name, int activate);
```

Arguments

name
The SEAPI name of the api partner to connect to.

activate
Reserved.

Return Values

SEA_OK
Successfully connected to the given api partner.

SEA_SRVDOWN
API Partner was not active.

Description

After connect is called, the given partner is entered into the internal partner list of the api, which will result in SEA_RQ_PARTNER_UP or SEA_RQ_PARTNER_DOWN requests being generated (regardless of whether the partner was active at connect or not).

Example

None.

Notes

- SEAConnect is intended for used by specialized api clients using the SEA_FN_MANUAL mask to SEAOpen.
- This function is new in SEAPI version 2.

SEAElcTimeToSEATime

Convert an Elcbas-style string time to SEAPI time format.

Synopsis

```
SEAStatus SEAElcTimeToSEATime (char *elctime,
                               SEATime *seatime);
```

Arguments

elctime
The time in Elcbas-style string format. This should be 17 characters long, in the format YYYYMMDDHHMMSSmmm (year, month, day, hour, minute, second, milliseconds, e.g. 20030901120000000 For 12 noon on the 1. September 2003.

seatime
The same time in SEAPI struct SEATime format.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
The seatime pointer is invalid, the elctime pointer is invalid, or the length of the elctime string is less than 17 characters.

Description

This function will convert a string in Elcbas style time format to the SEAPI format. Note that minimal checking is done on the format of this string except for its length.

Example

None.

Notes

- This function was new in SEAPI version 2.

SEAFree

Release data structures.

Synopsis

```
SEAStatus SEAFree (SEAHandle *data);
```

Arguments

data
A pointer to the handle for the data structure to deallocate. Upon successful deallocation this is set to SEA_NULL_HANDLE.

Return Values

SEA_OK
Normal, successful return

SEA_ILLHANDLE
The handle did not refer to a valid data structure

Description

This function may be used to deallocate a handle-based data structure for the API. Note that handles may be reused without deallocating (e.g. in the SEAGetRequest call, as the API will check the current value of the handle, and deallocate any unnecessary memory for each call.

Example

```
if (SEAFree (list) != SEA_OK)
    fprintf (stderr, “Error during deallocation\n”);
```

Notes

None

SEAGetPartner

Get the API name of the program that sent the current message.

Synopsis

```
SEAStatus SEAGetPartner (SEAHandle reqinfo,
                         char **partner_name);
```

Arguments

reqinfo
The handle to the message, as returned by SEAGetRequest

partner_name
Pointer to a char * that will be set to point to the SEAPI partner name. On return, the pointer will point to data within the API, which should not be modified. If the data is needed beyond the scope of the provided handle, the application should make a copy.

Return Value

SEA_OK
Normal, successful return.

SEA_ILLHANDLE
The handle did not refer to a valid, inbound message.

Description

This function will retrieve the API name of the application that originated the message referred to by the handle to the current application. The function may be useful e.g. in conjunction with the SEA_OP_MONITOR option. to find out whether the initiator or responder was started or stopped.

Example

None

Notes

- This function was new in SEAPI version 2.

SEAGetRequest

Get the next request from the API.

Synopsis

```
SEARequest SEAGetRequest (SEARequest reqMask,
                          int timeOut, SEAHandle *info);
```

Arguments

reqMask
Intended to provide filtering of requests, but currently ignored. Supply SEA_RQ_ALL for consistent behavior if a future version implements filtering.

timeOut
The time-out (in seconds) to wait for a new request. A value of 0 will cause a poll without waiting; SEA_INFINITE (-1) will wait indefinitely.

info
A handle to request-specific data. These data may not be accessed directly, but e.g. through one of the Objlist functions. If the same handle is used in multiple calls, the API will manage the memory associated with the handle, so that the current data will be unavailable to the application after the next call. Note that the handle must be initialized with SEA_NULL_HANDLE prior to first use.

Return Values

The function returns the request/response code provided by the sender in the SEASendResponse/SEASendRequest/SEASendRequestTo call.

For a full list of request/response codes, see (tbs)

The following codes are generated by this function, and will not return any message in the handle:

SEA_RQ_TIMEOUT
No request was available within the specified timeout, or no message was available if polling (timeout 0).

SEA_RQ_SHUTDOWN
An internal error occurred which prevents further use of the API. The application may attempt to recover by calling SEAClose and SEAOpen again, or may need to restart.

Description

This function receives a message through the API and returns the request code, with the info handle pointer being updated to refer to the actual message.

Example

```
SEAHandle info = SEA_NULL_HANDLE;
SEARequest rqid;

rqid = SEAGetRequest (SEA_RQ_ALL, SEA_INFINITE, &info);

switch (rqid)
   .
   .
```

Notes

- If polling (using a time-out of 0), the application must ensure that the function is called at least every few seconds, to avoid time-outs at the protocol level.
- As indicated under SEAOpen, this function is private to a thread under Win32; it will only return the events from the SEAOpen called for this thread.
- The requests are described in detail below. The following table describes the relationship between functions and options to SEAOpen and returned requests (tbd: table to be updated)

SEA_FN_REQUESTED	SEA_RQ_REQUESTED
SEA_FN_PERIODIC	SEA_RQ_PERIODIC
SEA_FN_UNSOLICITED	SEA_RQ_START_UNSOLICITED or SEA_RQ_STOP_UNSOLICITED
SEA_FN_COMMAND	SEA_RQ_COMMAND or SEA_RQ_SETPOINT
SEA_OP_CONFIG	SEA_RQ_CHECK_CONFIG
SEA_OP_COMMS	SEA_RQ_COMMS_EVENT
(always in case of error or stop of API)	SEA_RQ_SHUTDOWN
(depending on timeOut parameter)	SEA_RQ_TIMEOUT
(after call(s) to SEAAddInput)	SEA_RQ_USER_EVENT

SEAGetResult

Read the application-specified result code from an object list header.

Synopsis

```
SEAStatus SEAGetResult (SEAHandle info, SEAResult *res);
```

Arguments

info
The handle to the object list the result is stored in.

res
A pointer to a variable which will receive the result code.

Return Values

SEA_OK
Normal, successful return.

SEA_ILLHANDLE
The handle does not refer to an object list.

Description

This function is currently used only by the servers, and retrieves the result code stored with an object list.

Example

None

Notes

- This function was new in SEAPI version 3.

SEAGetSequence

Get the message sequence number.

Synopsis

```
SEAStatus SEAGetSequence (SEAHandle info, unsigned int *seq);
```

Arguments

info
The message for which the sequence number is wanted

seq
A pointer to a variable which will return the sequence number

Return Values

SEA_OK
Normal, successful return.

SEA_ILLHANDLE
The handle did not refer to a valid message.

Description

This function will return the message sequence number, which is set by SEASendRequest/SEASendRequestTo. This number may be used to correlate responses received with responses sent, by reading the number from the handle after it has been sent, and then reading the number of inbound responses.

Example

None.

Notes

- This function was new in SEAPI version 3.

SEAGetTime

Get the current time (UTC) in SEAPI time format.

Synopsis

```
SEAStatus SEAGetTime (SEATime *timebuf);
```

Arguments

timebuf
Pointer to a SEATime struct (allocated by caller) which will receive the current time as UTC.

Return Values

SEA_OK
Normal, successful return.

SEA_BADPARAM
The timebuf pointer is invalid.

Description

This function will get the current system time, including milliseconds, in UTC and return it in the SEAPI time format.

Example

None

Notes

- This function was new in SEAPI version 2.

SEALogMessage

Send application-specific text to the API log handler.

Synopsis

```
void SEALogMessage (int level, const char *fmt, ...);
```

Arguments

level

The log level to log this message at. This can be set using the SEASetLogLevel function, with the deafult level being SEA_LOG_INFO. One of:

SEA_LOG_DBG_1	Debug/Trace level (less verbose)
SEA_LOG_DBG_2	Debug/Trace level (more verbose)
SEA_LOG_DBG_3	Debug/Trace level (most verbose)
SEA_LOG_DATA	Used to log data being transferred
SEA_LOG_INFO	Informational level (during normal runtime)
SEA_LOG_WARNING	Warning level
SEA_LOG_ERROR	Error level
SEA_LOG_FATAL	Fatal error level

Fmt

A printf style format string, with arguments.

Return Values

None

Description

This currently prints a formatted message, with time information, on the standard error device (stderr).

Example

```
...
/* We got an error from a call to an external application */
SEALogMessage (SEA_LEV_ERROR, “Error from system %d”,
errorCode);
...
```

Notes

SEAManageInfoGet

Return a manage info structure from the current message.

Synopsis

```
SEAStatus SEAManageInfoGet (SEAHandle info,
                            SEAManageInfo **mip);
```

Arguments

info
The handle for the incoming message.

mip
A pointer to a SEAManageInfo pointer which will point to the manage info data in the message.

Return Values

SEA_OK
Normal, successful return

SEA_BADPARAM
The mip pointer is invalid

SEA_ILLHANDLE
The info handle does not refer to a valid manage info message

Description

This function is used to retrieve message data for messages with request code SEA_RQ_MANAGE or response code SEA_RS_MANAGE_RESPONSE.

The pointer returned points to data inside the API, which will be valid through the scope of the handle. If the data should be modified, a copy of the struct must be taken, and the handle updated with SEAManageInfoSet.

Example

None

Notes

- This function was new in SEAPI Version 3.
- The management interface is used between the Configuration tool and the servers, and is not intended for general use in this version.

SEAManageInfoNew

Allocate a message buffer for manage info use.

Synopsis

```
SEAStatus SEAManageInfoNew (SEAHandle *info,
                            SEAManageInfo *mip);
```

Arguments

info
A pointer to a handle which will be updated to refer to the new message. Any data previously associated with the handle will be released.

mip
A pointer to the SEAManageInfo structure to be used to initialize the message. May be NULL; the information may be set later with SEAManageInfoSet.

Return Values

SEA_OK
Normal, successful return

SEA_ILLHANDLE
The info handle is invalid

SEA_NOMEM
No more handles or no memory available to complete the operation

Description

This function is used to allocate a message for use in the SEAPI management interface. Optionally, the message is initialized with data from the SEAManageInfo struct.

Example

None

Notes

- This function was new in SEAPI Version 3.
- The management interface is used between the Configuration tool and the servers, and is not intended for general use in this version.

SEAManageInfoSet

Update a manage info message buffer with new data.

Synopsis

```
SEAStatus SEAManageInfoSet (SEAHandle info,
                            SEAManageInfo *mip);
```

Arguments

info
The handle for the message.

mip
A pointer to the SEAManageInfo structure to be used to initialize the message.

Return Values

SEA_OK
Normal, successful return

SEA_ILLHANDLE
The handle does not refer to a valid manage info message

SEA_BADPARAM
The mip pointer is invalid

Description

This function is used to update the data in a manage info message. It is typically used when creating a response to a SEA_RQ_MANAGE request.

Example

None.

Notes

- This function was new in SEAPI Version 3.
- The management interface is used between the Configuration tool and the servers, and is not intended for general use in this version.

SEANextPeriod

Increase the period counter in an object list.

Synopsis

```
SEAStatus SEANextPeriod (SEAHandle list);
```

Arguments

list
The handle to the object list.

Return Values

SEA_OK
Normal, successful return

SEA_ILLHANDLE
The handle does not refer to a valid object list

SEA_NOTIME
The object list does not contain a time specification

SEA_ENDPER
The number of periods specified is reached

Description

This is a convenience function that can be used in a responder API client to increase the period counter when responding to requests for time series data.

Example

See the t_rs_data sample source code.

Notes

- This function was new in SEAPI Version 3.

SEAObjlistAdd

Add an object to an object list

Synopsis

```
SEAStatus SEAObjlistAdd (SEAHandle list, SEAObject *object,
                         SEAValue *value);
```

Arguments

list
The handle to the object list.

object
The pointer to the object struct defining the object to add.

value
The pointer to the value for the object to add (may be NULL).

Return Values

SEA_OK
Normal, successful return

SEA_NOMEM
No memory available for operation

SEA_LISTFULL
The list is full (max 255 objects)

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function may be used to add objects to an already existing object list.

Example

```
SEAHandle list = SEA_NULL_HANDLE;
SEAObject object;
SEAValue value;

if (SEAObjlistNew (&list, 0) != SEA_OK)
  fprintf (stderr, “Failed to create object list\n”);
if (SEAObjlistAdd (list, &object, &value) != SEA_OK)
  fprintf (stderr, “Failed to add objects to list\n”);
```

Notes

- The data for object and value (if supplied) is copied, so the application may reuse/deallocate these after the call.

SEAObjlistAddEx

Add an object to an object list

Synopsis

```
SEAStatus SEAObjlistAddEx (SEAHandle list,
                           SEAObjectEx *object,
                           SEAValue *value);
```

Arguments

list
The handle to the object list.

object
The pointer to the extended object struct defining the object to add.

value
The pointer to the value for the object to add (may be NULL).

Return Values

SEA_OK
Normal, successful return

SEA_NOMEM
No memory available for operation

SEA_LISTFULL
The list is full (max 255 objects)

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function may be used to add objects to an already existing object list.

Example

```
SEAHandle list = SEA_NULL_HANDLE;
SEAObjectEx object;
SEAValue value;

if (SEAObjlistNew (&list, 0) != SEA_OK)
  fprintf (stderr, "Failed to create object list\n");
if (SEAObjlistAdd (list, &object, &value) != SEA_OK)
  fprintf (stderr, "Failed to add objects to list\n");
```

Notes

- The data for object and value (if supplied) is copied, so the application may reuse/deallocate these after the call.

SEAObjlistGet

Get the specified object from an object list. Also sets the current position of the list.

Synopsis

```
SEAStatus SEAObjlistGet (SEAHandle list, int position,
                         SEAObject **objptr);
```

Arguments

list
The object list we are reading.

position
The position (starting with 1) in the list we want read. The constants SEA_POS_CURRENT (return the current position) and SEA_POS_LAST (return the last position) are also defined.

objptr
The object structure pointer which will receive the object data. The returned pointer points to API data structures, and should be considered read-only. It will be value as long as the handle is in scope.

Return Values

SEA_OK
Normal, successful return

SEA_ILLPOS
The supplied position does not exist in the list

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function is similar to SEAObjlistNext, except it reads from the supplied position, also setting the current position of the list to this.

Example

```
SEAStatus status = SEA_OK;
SEAObject *optr;
int i;

status = SEAObjlistGet (list, 1, &optr);
for (i = 2; status == SEA_OK; i++)
```

```
{
    printf (“Name %s\n”, optr->name);
    status = SEAObjlistGet (list, i, &optr);
}
```

Notes

None

SEAObjlistGetEx

Get the specified object from an object list. Also sets the current position of the list.

Synopsis

```
SEAStatus SEAObjlistGetEx (SEAHandle list, int position,
                           SEAObjectEx **objptr);
```

Arguments

list
The object list we are reading.

position
The position (starting with 1) in the list we want read. The constants SEA_POS_CURRENT (return the current position) and SEA_POS_LAST (return the last position) are also defined.

objptr
The object structure pointer which will receive the object data. The returned pointer points to API data structures, and should be considered read-only. It will be value as long as the handle is in scope.

Return Values

SEA_OK
Normal, successful return

SEA_ILLPOS
The supplied position does not exist in the list

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function is similar to SEAObjlistNext, except it reads from the supplied position, also setting the current position of the list to this.

Example

```
SEAStatus status = SEA_OK;
SEAObjectEx *optr;
int i;

status = SEAObjlistGet (list, 1, &optr);
for (i = 2; status == SEA_OK; i++)
```

```
{
    printf (“Name %s\n”, optr->name);
    status = SEAObjlistGet (list, i, &optr);
}
```

Notes

None

SEAObjlistGetHeader

Return the header for the object list

Synopsis

```
SEAStatus SEAObjlistGetHeader (SEAHandle list,
                               SEAObjlistHeader **headptr);
```

Arguments

list
The object list we want to read the header for.

headptr
The address of a SEAObjlistHeader pointer. This will return a pointer into API data structures, and should not be used to update data. It will be valid as long as the handle is in scope.

Return Value

SEA_OK
Normal, successful return

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function is used to read the header of an objlist. The header contains information relating to the use of the objlist, such as transfer parameters.

Example

```
SEAObjlistHeader myHdr;
if (SEAObjlistGetHeader (list, &myHdr) != SEA_OK)
{
    fprintf (stderr, “Failed to read the list header\n”);
}
```

Notes

- The pointer will be set to point to a structure that is internal to the API. This should be considered read-only, and is only valid as long as the objlist in question exists.

SEAObjlistGetValue

Read the value for an object in an object list.

Synopsis

```
SEAStatus SEAObjlistGetValue (SEAHandle list, int position,
                              SEAValue **valptr);
```

Arguments

list
The object list we are dealing with.

position
The position in the list, typically SEA_POS_CURRENT.

valptr
The pointer to the value in the list. Note: This is read-only.

Return Values

SEA_OK
Normal, successful return

SEA_ILLPOS
The supplied position does not exist in the list

SEA_NOVAL
The object in the list did not have a value set

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function is used to read the value for an object in an object list, typically when receiving data, commands or setpoints. It returns a pointer to the value in the list, which should not be modified by the caller.

Example

```
SEAStatus status;
SEAObject *optr;
SEAValue *valptr;

if ((SEAObjlistGet (list, 1, &optr) != SEA_OK)
   ||(SEAObjlistGetValue (list, 1, &valptr) != SEA_OK))
{
  fprintf (stderr, “Failed to read object or value\n”);
```

```
}
```

Notes

- This call returns a pointer to the existing value information in an object list, which is read-only information, valid until the next call to this function or until the associated handle is reused. To update the value information, use SEAObjlistSetValue.

SEAObjlistNew

Create a new objlist, optionally populating it with one or more objects.

Synopsis

```
SEAStatus SEAObjlistNew (SEAHandle *list,
                         unsigned int size, ...);
```

Arguments

list
: A handle pointer. This should be for a valid handle. (If a non-empty handle, the previous contents are lost and memory reclaimed).

size
: The initial size of the list. A corresponding set of initializers should be supplied in the varargs part. These should be pointers to SEAObject and SEAValue structures, the contents of which will be copied into the list (a NULL pointer may be supplied for the value struct).

Return Values

SEA_OK
: Normal, successful return

SEA_NOMEM
: No memory available for operation

SEA_ILLSIZE
: The list is greater than supported maximum (255)

Description

This function will allocate an object list, initializing it with objects and values as required. The most typical use is for returning unsolicited data.

Example

```
SEAObject object;
SEAValue value;
SEAHandle list = SEA_NULL_HANDLE;

if (SEAObjlistNew (&list, 1, &object, &value) != SEA_OK)
{
  fprintf (stderr, “Failed to create objlist\n”);
}
```

Notes

- The input structures are copied, so the application may do as it pleases with those after the call.

SEAObjlistNewEx

Create a new objlist, optionally populating it with one or more objects.

Synopsis

```
SEAStatus SEAObjlistNew (SEAHandle *list,
                         unsigned int size, ...);
```

Arguments

list
: A handle pointer. This should be for a valid handle. (If a non-empty handle, the previous contents are lost and memory reclaimed).

size
: The initial size of the list. A corresponding set of initializers should be supplied in the varargs part. These should be pointers to SEAObjectEx and SEAValue structures, the contents of which will be copied into the list (a NULL pointer may be supplied for the value struct).

Return Values

SEA_OK
: Normal, successful return

SEA_NOMEM
: No memory available for operation

SEA_ILLSIZE
: The list is greater than supported maximum (255)

Description

This function will allocate an object list, initializing it with objects and values as required. The most typical use is for returning unsolicited data.

Example

```
SEAObjectEx object;
SEAValue value;
SEAHandle list = SEA_NULL_HANDLE;

if (SEAObjlistNew (&list, 1, &object, &value) != SEA_OK)
{
  fprintf (stderr, “Failed to create objlist\n”);
}
```

Notes

- The input structures are copied, so the application may do as it pleases with those after the call.

SEAObjlistNext

Return the next object in an object list.

Synopsis

```
SEAStatus SEAObjlistNext (SEAHandle list,
                          SEAObject **objptr);
```

Arguments

list
The object list we are reading

objptr
The object struct pointer for the next object on the list

Return Values

SEA_OK
Normal, successful return

SEA_ENDLIST
No more objects in list

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function will return the next object from an object list. On a list returned from SEAGetRequest the first call to this function will return the first object in the list. The SEAObjlistReset function can be used to reset a list to the first object.

Example

```
SEAStatus status;
SEAObject *optr;

while ((status = SEAObjlistNext (list, &optr)) == SEA_OK)
{
  printf (“Name %s\n”, optr->name);
}
```

Notes

- The caller should not modify the SEAObject structure returned.

SEAObjlistNextEx

Return the next object in an object list.

Synopsis

```
SEAStatus SEAObjlistNextEx (SEAHandle list,
                            SEAObjectEx **objptr);
```

Arguments

list
The object list we are reading

objptr
The object struct pointer for the next object on the list

Return Values

SEA_OK
Normal, successful return

SEA_ENDLIST
No more objects in list

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function will return the next object from an object list. On a list returned from SEAGetRequest the first call to this function will return the first object in the list. The SEAObjlistReset function can be used to reset a list to the first object.

Example

```
SEAStatus status;
SEAObjectEx *optr;

while ((status = SEAObjlistNextEx (list, &optr)) == SEA_OK)
{
  printf (“Name %s\n”, optr->name);
}
```

Notes

- The caller should not modify the SEAObject structure returned.

SEAObjlistReset

Reset the current item pointer for an objlist.

Synopsis

```
SEAStatus SEAObjlistReset (SEAHandle list);
```

Arguments

list
The handle to the object list.

Return Values

SEA_OK
Normal, successful return

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function is used to reset the internal position of an object list, so that SEAObjlistNext will start from the first object again. When a list is received from SEAGetRequest, this call is not needed for the first loop.

Example

See the t_rs_data sample code.

Notes

- This function was new in SEAPI Version 3.

SEAObjlistSetHeader

Update the object list header.

Synopsis

```
SEAStatus SEAObjlistSetHeader (SEAHandle list,
                               SEAObjlistHeader *hdr);
```

Arguments

list
The object list handle

hdr
A pointer to an SEAObjlistHeader structure which contains the new values for the header. The size field is not updated by this call, but is maintained internally to reflect the number of objects in the list.

Return Value

SEA_OK
Normal, successful return

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function cane be used to update the object header for an object list.

Example

None.

Notes

- This function was new in SEAPI Version 3.

SEAObjlistSetValue

Set the value for an object in an object list.

Synopsis

```
SEAStatus SEAObjlistSetValue (SEAHandle list, int position,
                              SEAValue *value);
```

Arguments

list
The object list containing the object we are setting the value for

position
The position in the list (1 is first). Usually this is the constant SEA_POS_CURRENT (=0) for the current position.

value
The value to set, supplied in a SEAValue structure.

Return Values

SEA_OK
Normal, successful return

SEA_ILLPOS
The supplied position does not exist in the list

SEA_ILLHANDLE
The handle did not refer to a valid object list

Description

This function is usually used to supply a value for an object when responding to various data transfer requests.

Example

```
SEAStatus status;
SEAObject *optr;
SEAValue value;

while ((status = SEAObjlistNext (list, &optr)) == SEA_OK) {
  if (optr->dtype == SEA_DT_FLOAT) {
    value.quality = SEA_Q_OK;
    value.v.fval = 10.0;
  }
  else {
```

```
    value.v.fval = 0.0;
    value.quality = SEA_Q_NOTOK;
  }
  if (SEAObjlistSetValue (list, SEA_POS_CURRENT, &value)
      != SEA_OK) {
    fprintf (stderr, “Error setting value\n”);
  }
}
```

Notes

- The information in the value struct is copied, thus it is the responsibility of the caller to deallocate the original structure.

SEAOpen

The SEAOpen function Initializes the SEAPI. It should be called prior to calling any other function in the API.

Synopsis

```
SEAStatus SEAOpen (const char *appName,
                   SEAFunction functionMask,
                   SEAOption optionMask)
```

Arguments

appName

The name of this interface application (used for identification purposes). Null-terminated string. The string is significant to 8 characters, which must be unique on a system.

functionMask

A bitmask specifying which functions will be supported by the API, formed by OR-ing together from the following list:

SEA_FN_UNSOLICITED	Handle unsolicited data transfer	V1/V2 Responder client
SEA_FN_PERIODIC	Handle Periodic data transfer	V1/V2 Responder client
SEA_FN_REQUESTED	Handle Requested data transfer	V1/V2 Responder client
SEA_FN_COMMAND	Handle Commands and setpoints	V1/V2 Responder client
SEA_FN_AUTO	Connect to initiator (V2). Both servers (V3)	V2 Initiator client, V3 Initiator/Responder client
SEA_FN_INCLIENT	Connect to initiator	V3 Initiator client
SEA_FN_RSCLIENT	Connect to responder	V3 Responder client

The V1 responder functions implies creation of ‘built-in’ clients, and can only be used by one client (for each function) at a time in a system.

optionMask

A bitmask specifying additional options, formed by OR-ing together from the following list:

SEA_OP_CONFIG	Defined, but not used
SEA_OP_COMMS	Defined, but not used.
SEA_OP_EXCEPTION	Defined, but not used
SEA_OP_MONITOR	Receive notification when associated server(s) starts and stops. (new in version 3).

Return Values

SEA_OK
Normal, successful return

SEA_INUSE
The specified name is already used by another process, or the specified V1 responder function is already used by another process.

Description

The SEAOpen call must be called prior to calling any other functions in the API. It will initialize the API, and set up the event handler according to the specified function mask.

Example

```
if (SEAOpen (“MYAPP”, SEA_FN_AUTO, 0)!= SEA_OK)
{
  fprintf (stderr, “Failed to initialize SEAPI\n”);
  exit (2);
}
```

Notes

- For the Win32 platform the SEAPI utilizes thread-local storage, so that multiple threads within a process may access the API simultaneously, each thread calling SEAOpen as required. All other API functions must then be called from the same thread.
- The function is designed to succeed even if the local API processes are not yet started. The application should wait in or poll SEAGetRequest to get such data when available.

SEARemoveInput

Remove user-supplied event sources from SEAGetRequest

Synopsis

```
SEAStatus SEARemoveInput (SEAInput input);
```

Arguments

input
The event source to remove. See SEAAddInput.

Return Values

SEA_OK
Normal, successful return

SEA_ILLINP
The input source is invalid (no such input)

Description

This function is used to reverse the effect of SEAAddInput

Example

None

Notes

None

SEASendRequest

Send a request to an implicit server.

Synopsis

```
SEAStatus SEASendRequest (SEARequest request,
                          SEAHandle data);
```

Arguments

request
The request to be sent to the API.

data
The data (e.g. object list) associated with the request.

Return Values

SEA_OK
Normal, successful return

SEA_SRVDOWN
The server/partner process was not running

Description

The SEASendRequest function sends a message to an implied server. New applications should use the explicit SEASendRequestTo instead.

For V1 clients, this will send a request to the responder.

For V2 and newer clients, this will send a request to the initiator.

Example

None.

Notes

- The SEASendRequestTo function is the preferred alternative for newer clients.

SEASendRequestTo

Send a request to a specified API partner (server).

Synopsis

```
SEAStatus SEASendRequestTo (const char *srv,
                            SEARequest request,
                            SEAHandle data);
```

Arguments

srv
The API name for the partner/client this message is to be sent to. The header file defines SEA_SRV_IN for the initiator name and SEA_SRV_RS for the responder name.

request
The request code to send.

data
The handle to the associated data, e.g. object list.

Return Values

SEA_OK
Normal, successful return

SEA_SRVDOWN
The server/partner process was not running

Description

This function sends a request message to the indicated API partner (server).

Example

See the t_in_cmd sample.

Notes

- This function was new in version 2.

SEASendResponse

Send a response to the message originator

Synopsis

```
SEAStatus SEASendResponse (SEAResponse response,
                           SEAHandle data);
```

Arguments

response

The response id for this data transfer. Currently this mat be one of the following (list to be reviewed)

SEA_RS_DATA	This is a response to a requested or periodic data transfer request
SEA_RS_UNSOLICITED_STARTED	This is a response to a start unsolicited request
SEA_RS_UNSOLICITED	This is unsolicited data
SEA_RS_COMMAND_RESPONSE	This is a response to a command
SEA_RS_SETPOINT_RESPONSE	This is a response to a setpoint request
SEA_RS_CONFIG_CHECKED	The configuration for an objlist has been checked
SEA_RS_ERROR_RESPONSE	This indicates that the last request was unacceptable
SEA_RS_DONTCARE	This indicates that there is no particular response to a request

For a description of the correlation between requests and responses, see the request descriptions later in this chapter.

data

This is a handle referring to data relevant to the response. Often this will be the same handle as received by SEAGetRequest

Return Values

SEA_OK

Normal, successful return

SEA_SRVDOWN

The server/partner process was not running

Description

This function is used to return data and other responses from the API client application to the originator of the request message.

Example

```
SEAHandle objlist = SEA_NULL_HANDLE;
...
for (;;)
  {
    SEAGetRequest (SEA_RQ_REQUESTED, SEA_INFINTE, &objlist);
    /* Retrieve data into list */
    SEASendResponse (SEA_RS_DATA, objlist);
  }
```

Notes

- For a description of the relationship between requests and responses, see below.

SEASetLogLevel

Sets the library log level.

Synopsis

```
void SEASetLogLevel (int level);
```

Arguments

level
The log level to set (see SEALogMessage for a list of levels)..

Return Values

None.

Description

This function is used to set the library log level. When logging a message with SEALogMessage, only messages with a level equal or higher than the set level will actually be output.

The default log level is SEA_LOG_INFO.

Example

- None.

Notes

SEASetLogTarget

Set the target for the library log.

Synopsis

```
void SEASetLogTarget (int type, const char* logName);
```

Arguments

type
The type of logging target, one of the constants

- LOG_FILE – log to file
- LOG_CATEGORY – log using the log4cpp library

logName
The log target, a file if type is LOG_FILE, otherwise a log4cpp category name.

Return Values

None.

Description

This function is used to set the target for the library logging function. The default is to log to the standard error output.

Note that the use of the log4cpp library is primarily intended for use internally by the Elcbas servers, as it requires a priori initalization to be useful.

Example

None.

Notes

SEASetResult

Update the result code in the object list header.

Synopsis

```
SEAStatus SEASetResult (SEAHandle list, SEAResult res);
```

Arguments

list
The object list to set the result code in.

res
The result code to set.

Return Values

SEA_OK
Normal, successful return.

SEA_ILLHANDLE
The handle does not refer to an object list.

Description

This function is used to set a result code in an object list. Currently, the only supported use of this function is when handling requested data transfer in responder API clients.

Example

None.

Notes

- This function was new in SEAPI Version 3.

SEASystemTimeToSEATime

Convert a Win32 SYSTEMTIME structure to SEAPI time format.

Synopsis

```
SEAStatus SEASystemTimeToSEATime (SYSTEMTIME *systemtime,
                                  SEATime *seatime);
```

Arguments

systemtime
Pointer to a SYSTEMTIME structure, as defined in the Win32 API.

seatime
Pointer to a SEATime structure which will be updated based on the contents of the SYSTEMTIME input.

Return Values

SEA_OK
Normal, successful completion.

SEA_BADPARAM
Either of the input pointers are invalid.

Description

This function performs a straight copy of corresponding fields between the two structures.

Example

None.

Notes

- This function was new in SEAPI Version 2.
- This function is only available on the Windows platform.
- No checks are performed on the validity of the contents of the input structure.

SEATimeToAlibTime

Convert time from SEAPI format to Elcom alib format (integer array) .

Synopsis

```
SEAStatus SEATimeToAlibTime (SEATime *seatime,
                             int altime[7]);
```

Arguments

seatime
[in] The input time in SEAPI format.

altime
[out] The same time in Elcom alib format, ie. an integer array, with year as year – 1900.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
The seatime pointer is invalid.

Description

This function will convert a time in the SEAPI format to the format used by the Elcom-90 API (alib).

Example

None available.

Notes

- This function is new in SEAPI version 3.
- No validity checks are performed on the input time.

SEATimeToElcTime

Convert time from SEAPI format to Elcbas-style string format.

Synopsis

```
SEAStatus SEATimeToElcTime (SEATime *seatime, char *elctime);
```

Arguments

seatime
[in] The input time in SEAPI format.

elctime
[out] The same time in Elcbas-style string format.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
One or both pointers are invalid.

Description

This function creates a string in the format YYYYMMDDHHMMSSmmm from the input format, e.g. 20030915120000000.

Example

None.

Notes

- This function is new in SEAPI version 2.
- No validity checks are performed on the input time.

SEATimeToSystemTime

Convert time from SEAPI format to Win32 SYSTEMTIME format.

Synopsis

```
SEAStatus SEATimeToSystemTime (SEATime *seatime,
                               SYSTEMTIME *systemtime);
```

Arguments

seatime
[in] The input time in SEAPI format.

systemtime
[out] The same time in Win32 SYSTEMTIME format.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
One or both pointers are invalid.

Description

This function converts a time in SEAPI format to the Win32 SYSTEMTIME format. The function is a straight copy, except for the weekday field in SYSTEMTIME (wDayOfWeek), which is computed.

Example

None.

Notes

- This function was new in SEAPI Version 2.
- This function is only available on the Windows platform.
- No checks are performed on the validity of the contents of the input structure.

SEATimeToTimeVal

Convert time from SEAPI format to a Unix timeval structure.

Synopsis

```
SEAStatus SEATimeToTimeVal (SEATime *seatime,
                            struct timeval *tv);
```

Arguments

seatime
[in] The input time in SEAPI format.

tv
[out] The same time in Unix timeval format.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
One or both pointers are invalid.

Description

This function converts the input time in SEATime format to a struct timeval, which contains the elapsed seconds since 1.1.1970 together with the elapsed microseconds within the second. The input time is assumed to be in UTC.

Example

None.

Notes

- This function was new in SEAPI Version 2.
- This function is only available on the Unix platforms.
- No checks are performed on the validity of the contents of the input structure.

SEATimeToTm

Convert time from SEAPI format to a struct tm format.

Synopsis

```
SEAStatus SEATimeToTm (SEATime *seatime, struct tm *tmp);
```

Arguments

seatime
[in] The input time in SEAPI format.

tmp
[out] The same time in struct tm format, as defined in ANSI C.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
One or both pointers are invalid.

Description

This function converts from the SEAPI time format to a struct tm, as defined in ANSI C. Only corresponding fields are converted, and milliseconds are ignored.

Example

None.

Notes

- This function was new in SEAPI Version 2.
- No checks are performed on the validity of the contents of the input structure.

SEATimeValToSEATime

Convert from Unix timeval structure to SEAPI time format.

Synopsis

```
SEAStatus SEATimeValToSEATime (struct timeval *tv,
                               SEATime *seatime);
```

Arguments

tv
[in] The time in struct timeval format.

seatime
[out] The same time in SEAPI time format.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
One or both pointers are invalid.

Description

This function converts the supplied timeval structure (as UTC) to a corresponding SEATime structure. The microseconds are truncated to milliseconds.

Example

None.

Notes

- This function was new in SEAPI Version 2.
- This function is only available on the Unix platforms.
- No checks are performed on the validity of the contents of the input structure.

SEATmToSEATime

Convert from struct tm format to SEAPI time format.

Synopsis

```
SEAStatus SEATmToSEATime (struct tm *tmp, SEATime *seatime);
```

Arguments

tmp
[out] The input time in struct tm format, as defined in ANSI C.

seatime
[in] The same time in SEAPI format.

Return Values

SEA_OK
The operation completed successfully.

SEA_BADPARAM
One or both pointers are invalid.

Description

This function converts the input struct tm time, as defined in ANSI C, to SEAPI time format. Milliseconds are set to zero.

Example

None.

Notes

- This function was new in SEAPI Version 2.
- No checks are performed on the validity of the contents of the input structure.

3.2 Structures

Please refer to the header file seapub.h for structures.

3.3 Constants

Please refer to the header file seapub.h for constants.

3.4 Macros

Please refer to the header file seapub.h for macros.