E U R O P E A N S O U T H E R N O B S E R V A T O R Y

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

VLT Software

CCS/LCC - Common Access Interface

User Manual

Doc.No. VLT-MAN-ESO-17210-0719

Issue 1.3

Date 17/01/96

	Name	Date	Signature
Prepared	P.Duhoux	17/01/96	
Approved	B.Gustafsson, B.Gilli	17/01/96	
Released	G.Raffi	/01/96	

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
1.0/prep 1	29/09/94	All	First preparation
1.0	26/10/94	Section 1.1 Section 1.3 Section 2.2 Section 3 Section 4.3 Section 5	Version number & date Set date for Sepcification/Design document New interface description to functions Updated man pages Version display Updated error messages
1.1	13/01/95	Section 2.2 Section 2.3 Section 3	New Interface to caiGetFunctions() New Interface to caiGetAttributes() New functions: - caiRWTable() & caiRWVector() New man pages
1.2	22/06/95	All	Database access functions ported to CCS
1.3	28/09/95	Section 2.3.4 Section 3	Added function caiGetAttrInfo() Updated man-pages

TABLE OF CONTENTS

1 INTRODUCTION

The software described in this manual is intended to be used in the ESO VLT project by ESO and authorized external contractors only.

While every precaution has been taken in the development of the software and in the preparation of this document, ESO assumes no responsibility for errors or omissions, or for damage resulting from the use of the software or of the information contained herein.

1.1 Purpose

This document is the User Manual of the CCS/LCC - Common Access Interface (CAI), version 1.14.
The Common Access Interface is built on top of the database access functions provided by CCS and LCC, making direct accesses to dynamically resolved attributes easier and faster.
This interface provides also services for dynamic access of functions on LCU only.

It is intended to provide all the necessary information to use this software to develop application software running on LCU and WS.

The manual assumes that the reader has a good knowledge of UNIX, VxWorks, C language and is familiar with VxWorks and Unix development environments.

In addition to the **Introduction** section, this manual contains 4 major sections:

User's Guide, describing the functionality provided by the software, including examples of utilization.

Reference Manual, describing all the functions available to the applications and the application developers.

Installation Guide, describing how to install and make the software ready to be used.

Error Messages, providing the list of errors and diagnostic messages.

1.2 Scope

The CCS/LCC Common Access Interface Software provides an additional set of common functions to the CCS/LCC services and which shall be available on each WS/LCU that requires it.
This implementation consists of one single module (on LCU) / library (on WS):

- **cai** CCS/LCU Common Access Interface module
- **libcai.a** CCS/WS Common Access Interface library

The following environments are required to run and make the software described:

- L**CU** :
  - VxWorks Operating System version 5.2 or higher,
  - LCU Common Software DEC95 Release
- **WS**:
  - VLT Common Software DEC95 Release

1.3 Reference Documents

The following documents contain additional information and are referenced in the text:

[1] VLT-PRO-ESO-10000-0228, 1.0 10/03/93 --- VLT Software Programming Standards

[2] VLT-MAN-ESO-17210-0619, 1.3 17/08/95 --- CCS Central Common Software User Manual
[3] VLT-MAN-SBI-17210-0001, 2.5 19/01/96 --- LCU Common Software User Manual
[4] VLT-SPE-ESO-17210-0718, 1.2 21/06/95 --- CCS/LCC - Common Access Interface Functional Specification and Design Document
[5] VLT-MAN-ESO-17200-0642, 1.5 22/01/96 --- VLT Common Software Installation Manual
[6] VLT-MAN-ESO-17200-0981, 1.0 15/01/96 --- VLT Software Problem Report Change Request User Manual

1.4 Abbreviations and Acronyms

The following abbreviations and acronyms are used in this document:

API	Application Programmatic Interface
CAI	CCS/LCC Common Access Interface
CCS	Central Control Software
DB	On-Line Database
DCL	Device Control Library (see SDL)
HW	Hardware
LCC	LCU Common Software
LCU	Local Control Unit
SDL	Single Device Library (see DCL)
SW	Software
TBD	To Be Defined
VLT	Very Large Telescope
WS	WorkStation (Unix)

1.5 Stylistic Conventions

The following styles are used:

bold in the text, for commands, filenames, pre/suffixes as they have to be typed.

italic in the text, for parts that have to be substituted with the real content before typing.

`teletype` for examples.

<name> in the examples, for parts that have to be substituted with the real content before typing.

bold and *italic* are also used to highlight words.

1.6 Naming Conventions

This implementation follows the naming conventions outlined in [1].

1.7 Problem Reporting / Change Request

The procedure follows the recommendations stated in [6].

2 USER'S GUIDE

This chapter is a guide for the designer when implementing applications using the Common Access Interface functions.

2.1 Overview

2.1.1 Common Access Interface

The Common Access Interface provides a set of functions to support the VLT application software. These functions are based on CCS/LCC and provide in this extend additional services to them. The Common Access Interface functions are grouped in two categories:

- Dynamical interface to functions: for applications where the interface to function libraries shall be dynamic. (LCC only)
    a. resolution of entry points of functions from the global symbol table
    b. call facility to these functions.
- Direct access to database attributes: for applications where the access to database attributes shall be fast (direct access).
    a. resolution of direct addresses of attributes underlying a point
    b. access facility to these attributes

This section provides mainly implementation examples; the detailed reference to these functions can be found in the next section.

2.1.2 Availability

The Common Access Interface services are available as direct procedural call from any program. The following paragraphs describe the functions provided by the **cai** module and **libcai** library.

2.2 Interface to functions (LCC only)

This service applies merely for modules that interface driver software with API level. These SDL / DCL modules are exchangeable, with respect to the implementation but not to the functionality. The API level is then completely independent of the underlying software (and of the hardware).
Thus each SDL / DCL module, providing a specific set of functions, must, apart of the module name, assign the same names to the same functions.
Moreover, every function of a module that is to be accessed by the **cai** module must have the same programmatic interface:

```
ccsCOMPL_STAT modVFunction( void *desc, ccsERROR *error, va_list parList)
```

This function provides only the extraction of the expected parameters from the variable parameter list and the call to *modFunction()*.

```
#include <stdarg.h>
#include "mod.h"

ccsCOMPL_STAT modVFunction( void *desc, ccsERROR *error, va_list parList)
    {
    ccsCOMPL_STAT status;

    type1 arg1;            /* These are the three arguments expected   */
    type2 arg2;            /* by the function modFunction() in this    */
    type3 arg3;             /* order                                   */

    /*
     * use va_arg macro to extract the expected parameters from the
     * parList argument.
     */
    arg1 = va_arg( parList, type1 );
    arg2 = va_arg( parList, type2 );
    arg3 = va_arg( parList, type3 );

    /*
     * call requested SDL routine
     */
    status = modFunction( desc, arg1, arg2, arg3, error );

    return( status );
    }
```

2.2.1 Resolving the entries

For a given module, a set of functions has to be provided according to the specifications of this module. Each module is identified uniquely by its name. The list of the names of the functions (without the module's name) must be terminated with the NULL pointer. The address of the entry table is returned in a character pointer.

```
char *funcNames[] =
    {
    "GetStatus",          /* is the function modVGetStatus()       */
    "SetOpMode",          /* is the function modVSetOpMode()       */
    "Reset",              /* is the function modVReset()           */
    NULL                  /* MANDATORY : end of the list           */
    };
char *funcPrefix = "mod";
char *funcTable;

...

if (caiGetFunctions( &funcTable,funcPrefic,funcNames,
                     error ) == FAILURE)
    {
    /* process error */
    ...
    }
/* continue the process */
...
```

The possible errors are:

caiERR_PARAMETER	:	An input parameter is invalid: - funcTable is NULL, - funcNames is NULL or entry is empty (""), - list of valid function names is empty.
caiERR_MEMORY	:	Cannot allocate function entry table or list of function entries.
caiERR_SYMBOL	:	Cannot find symbol in global symbol table.

The entry table will be returned to the memory pool by the call to **caiFreeFuncTable**.

```
...
caiFreeFuncTable( funcTable );
...
```

2.2.2 Calling the functions

The functions are referenced by index in the function entry table. The variable list of parameters depends on the SDL / DCL function (see the User Manual of the concerned module for further information).

Assuming the example described above, one wants to invoke the function modGetStatus(), where the address of the status data structure and a boolean flag are to be passed as arguments.

The calling sequence is then coded as follows:

```
modDESCRIPTION modDesc;       /* associated descriptor data struct */
int            funcIndex = 0;  /* for GetStatus()                   */
modSTATUS      modStatus;      /* status data structure             */
vltLOGICAL     modFlag = TRUE; /* additional flag for modGetStatus()*/

...

if (caiCallFunction( funcTable,funcIndex,error,&modDesc,
                     &modStatus,modFlag ) == FAILURE)
    {
    /* process error */
    ...
    }
/* continue the process */
...
```

The possible errors are:

`caiERR_PARAMETER` : An input parameter is invalid:
- funcTable is NULL ,
- invalid table identifier,
- funcIndex is out of valid range.

`caiERR_ENTRY` : Function entry is NULL.

`caiERR_CALL_SDL` : An error occurred while calling the SDL function.

2.3 Interface to Database Direct Access (CCS and LCC)

2.3.1 Resolving the attributes' direct addresses

For a given database point, one wants to resolve the direct addresses of a set of attributes required to be accessed in a fast way. Every kind of attribute is allowed: single scalar element, vector or element of a vector, table or row of a table or field of a row.
It is also possible to refer attributes located under sub-points of the entry point (no limitation in depth); the attribute is then referenced by its relative symbolic path to the entry point.
For attributes underlying the entry point, the leading dot '**.**' may be omitted.
The list of the names of the attributes must be terminated with the NULL pointer. The address of the attribute table is returned in a character pointer.

```
dbSYMADDRESS point = ":dbRoot:applPoint";
char *attrNames[] =
    {
    "scalar",                 /* scalar of type dbUINT32           */
    "modPoint.scalar",        /* scalar of type dbUINT32           */
    "modPoint.vector",        /* vector of 10 dbINT16              */
    "modPoint.vector(3)",     /* - its fourth element              */
    "modPoint.table",         /* table of 10 records               */
    "modPoint.table(4)",      /* - its last record                 */
    "modPoint.table(2,1)",    /* - the 2nd field of the 3rd record */
                              /*   as a dbBYTES8, the first being  */
                              /*   of type dbINT8                  */
    NULL                      /* MANDATORY : end of the list       */
    };
char *attrTable;

...

if (caiGetAttributes( &attrTable,point,attrNames,error ) == FAILURE)
    {
    /* process error */
    ...
    }
/* continue the process */
...
```

The possible errors are:

`caiERR_POINT` : The database point is invalid.
`caiERR_PARAMETER` : An input parameter is invalid:
- attrNames is NULL or an attribute name is empty (""),
- list of valid attribute names is empty.

`caiERR_MEMORY` : Cannot allocate attribute address table or list of attribute addresses.
`caiERR_ATTRIBUTE` : Cannot find attribute under the given point.

The attributes table will be returned to the memory pool by the call to **caiFreeAttrTable**.

```
caiFreeAttrTable( attrTable );
```

2.3.2 Accessing the single attributes

The database items are referenced by index.
The function **caiRWAttribute()** allows the access of any attribute, as resolved by **caiGetAttributes()**. The access is performed as declared: e.g. reading the 5th entry of the example described above leads to read the complete table, whereas writing the 4th item leads to access only the fourth element of the vector.

Caution: Enough memory must be allocated to store the item.

Assuming the example described above, one wants to read the attribute 'scalar', being a 32-bit integer, and write the 2nd field of the 3rd record of the table (already resolved as an entry), being a 8-char string.

The calling sequence is then coded as follows:

```
vltINT32       scalar;          /* scalar value                  */
vltBYTES8      string;          /* a string ...                  */
...

/* read the scalar attribute 'scalar' */
if (caiRWAttribute( attrTable,FALSE,0,&scalar,error ) == FAILURE)
    {
    /* process error */
    ...
    }
/*
 * write the string in the 2nd field of the 3rd record of the table
 */
strcpy(string,"Hallo!");
if (caiRWAttribute( attrTable,TRUE,6,string,error ) == FAILURE)
    {
    /* process error */
    ...
    }
/* continue the process */
...
```

The possible errors are:

caiERR_PARAMETER : An input parameter is invalid:
- attrTable is NULL,
- invalid table identifier,
- attrIndex is out of declared range,
- attrBuffer is NULL,
- attribute address entry is NULL.

caiERR_READ : Database direct read failed.
caiERR_WRITE : Database direct write failed.

2.3.3 Accessing the structured attributes

The two functions **`caiRWTable()`** and **`caiRWVector()`** allow the access to part of attributes of type table or vector, resolved as a whole by **`caiGetAttributes()`**.

Caution: Enough memory must be allocated to store the item.

Assuming the example described above, one wants to read the 3rd to 7th elements of the attribute 'vector', being an array of 10 16-bit integers, and write the 1st field of the 3rd to 4th records of the table, being 8-bit integers.

The calling sequence is then coded as follows:

```
vltINT16       array[5];        /* array of scalars              */
vltINT8        integer[2];     /* array of 8-bit integers        */
...

/* read the 3-7 elements of the vector attribute 'vector' */
if (caiRWVector( attrTable,FALSE,2,&scalar,2,6,error ) == FAILURE)
    {
    /* process error */
    ...
    }
/*
 * write the string as the 1st field of the 3-4 records of the table
 */
integer[0] = -32; integer[1] = 127;
if (caiRWTable( attrTable,TRUE,4,integer,2,3,0,0,error ) == FAILURE)
    {
    /* process error */
    ...
    }
/* continue the process */
...
```

The possible errors are:

`caiERR_PARAMETER` : An input parameter is invalid:
- attrTable is NULL,
- invalid table identifier,
- attrIndex is out of declared range,
- attrBuffer is NULL,
- attribute address entry is NULL.

`caiERR_READ` : Database direct read failed.
`caiERR_WRITE` : Database direct write failed.

Addressing syntax: The two integers defining the accessed range must fulfill the following requirements: the range must fit the existing range, the last must be greater than or equal the first one, -1 means from the beginning, resp. until the end. For performance reasons, the indexes do not comply with the general addressing conventions of database attributes.
Caution: The data buffer is treated as a byte buffer, which may be subject to alignment. Proper function is granted when no alignment is made (4 bytes boundaries).

2.3.4 Getting the attribute's information

The function **caiGetAttrInfo()** returns the complete attribute's information, ie attribute's type, number of rows / elements, number of fields in row and data types.

Caution: Enough memory must be allocated to store the data types.

Assuming the example described above, one wants to retrieve the attribute information of the table.

The calling sequence is then coded as follows:

```
dbATTRTYPE attrType;          /* attribute type                  */
vltUINT8   fldCnt;            /* number of fields                */
vltUINT16  recCnt,recsUsed;   /* record number / records used    */
vltUINT32  recSize;           /* record size                     */
dbTYPE     dataType[dbMAX_FIELD_CNT+1]; /* data types             */
...

/* retrieve the attribute information of the table */
if (caiGetAttrInfo( attrTable,4,&attrType,&fldCnt,&recCnt,
                    &recSize,&recsUsed,dataType,error ) == FAILURE)
    {
    /* process error */
    ...
    }
printf("AttrType    = %d\n",attrType);
printf("Records     = %d \t Fields       = %d\n",recCnt,fldCnt);
printf("Record Size = %d \t Records Used = %d\n",recSize,recsUsed);
printf("DataTypes : \n");
for (i=0;i<=fldCnt;i++)
    printf("Field #%1d : Type = %d\n",i,dataType[i]);
```

The output will be:

```
AttrType    = 2                    => dbTABLE
Records     = 10     Fields       = 2
Record Size = 9      Records Used = 10
DataTypes :
    Field #0 : Type = 2            => dbINT8
    Field #1 : Type = 17           => dbBYTES8
    Field #2 : Type = 0            => dbUNDEFINED
```

The parameters may be omitted (passed as NULL), . The *dataType* array is terminated by the value dbUNDEFINED. For attributes of type dbSCALAR or dbVECTOR, the parameter *fldCnt* is set to 1. For attributes of type dbSCALAR, the parameter *recCnt* is set to 1.
The parameter *recsUsed* takes always the value of *recCnt*.
Warning: enough memory space must be allocated for the vector *dataType*.

The possible errors are:
caiERR_PARAMETER : An input parameter is invalid:
- attrTable is NULL,
- invalid table identifier,
- attrIndex is out of declared range

3 REFERENCE MANUAL

This chapter provides a detailed description of the Common Access Interface, namely functions and include file. Each function is described by its man page.

3.1 Interface to Functions

3.1.1 caiGetFunctions(3)

NAME

caiGetFunctions - get entry points for SDL routines

SYNOPSIS

```
#include "cai.h"

ccsCOMPL_STAT caiGetFunctions (
                                INOUT char      **table,
                                IN    vltBYTES8   prefix,
                                IN    char       *funcNames[],
                                OUT   ccsERROR   *error
                               )
```

DESCRIPTION

This routine resolves the addresse of the entry points of the SDL functions associated to a specified board.
The function names are built as follows <prefix>V<funcNames[index]>()
If no prefix is given, a lower case v is prefixed to the function's name.
The table <table> is dynamically allocated.
The entry points are found by name in the system symbol table.

table - address of pointer to function table memory space
prefix - SDL/DCL module name
funcNames - table of the names of the functions ending with NULL
error - error handling data structure

If anything goes wrong FAILURE is returned and the error reason and description is returned in the <error> variable.

RETURN VALUES

SUCCESS if everything ok
FAILURE if anything went wrong

- - - - - -
Last change: 21/06/95-07:48

3.1.2 caiCallFunction(3)

NAME

caiCallFunction - call SDL function via table

SYNOPSIS

```
#include "cai.h"

ccsCOMPL_STAT caiCallFunction (
                                IN    char     *table,
                                IN    int       caiNum,
                                OUT   ccsERROR *error,
                                IN    void     *desc,
                                INOUT ...
                               )
```

DESCRIPTION

This routine performs all SDL function calls. The parameters passed to this function depend on the requested SDL call.

```
table  - pointer to SDL function table
caiNum - number of requested SDL function
desc   - pointer to  descriptor data structure
error  - error handling data structure
...    - variable parameter list
```

RETURN VALUES

OK, if function terminated successfully
FAILURE, if something went wrong

```
- - - - - -
Last change:  21/06/95-07:48
```

3.1.3 caiFreeFuncTable(3)

NAME

caiFreeFuncTable - free memory space of functions table

SYNOPSIS

```
#include "cai.h"

void caiFreeFuncTable (
                        IN  char *table
                       )
```

DESCRIPTION

This routine returns to the system all the memory space pointed to by <table>.

- - - - - -

Last change: 21/06/95-07:48

3.2 Direct Database Access

3.2.1 caiGetAttributes(3)

NAME

caiGetAttributes - get direct addresses of attributes

SYNOPSIS

```
#include "cai.h"

ccsCOMPL_STAT caiGetAttributes (
                                 INOUT char         **table,
                                 IN    dbSYMADDRESS   point,
                                 IN    char          *attrNames[],
                                 OUT   ccsERROR      *error
                                )
```

DESCRIPTION

This routine resolves the direct addresses of the symbolic paths for the database attributes that require a fast access. The addresses are written into <table>. The table <table> is dynamically allocated.

```
table     - address of pointer to attribute table memory space
point     - database point name
attrNames - table of the names of the attributes ending with NULL
error     - error handling data structure
```

The item names can point to attributes located at any level below the point <point>.

If anything goes wrong FAILURE is returned and the error reason and description is returned in the <error> variable.

RETURN VALUES

SUCCESS if everything ok
FAILURE if anything went wrong

CAUTIONS

The behaviour is undefined if the entry point is “:”.

```
- - - - - -
Last change:  21/06/95-07:48
```

3.2.2 caiRWAttribute, caiReadAttribute, caiWriteAttribute(3)

NAME

caiRWAttribute,caiReadAttribute,caiWriteAttribute -
Direct R/W attributes in database

SYNOPSIS

```
#include "cai.h"

ccsCOMPL_STAT caiRWAttribute (
                               IN  char       *table,
                               IN  vltLOGICAL  write,
                               IN  int         attrNum,
                               IN  void       *attrBuffer,
                               OUT ccsERROR   *error
                              )
ccsCOMPL_STAT caiReadAttribute, caiWriteAttribute (
                               IN  char       *table,
                               IN  int         attrNum,
                               IN  void       *attrBuffer,
                               OUT ccsERROR   *error
                              )
```

DESCRIPTION

This routine read/write the attribute indexed by <attrNum> from/to the database. The addressing is direct, the address is read out of the table <table>. The attribute's value is pointed to by <attrBuffer>. The flag <write> selects the operation to perform.

```
table      - pointer to attribute table
write      - operation type (FALSE : read ; TRUE : write)
attrNum    - attribute's index in attribute table
attrBuffer - pointer to attribute's memory space
error      - error handling data structure
```

If anything goes wrong FAILURE is returned and the error reason and description is returned in the <error> variable.

RETURN VALUES

SUCCESS if everything ok
FAILURE if anything went wrong

CAUTION

It is the responsibility of the designer to provide enough memory space for reading the database attributes.

- - - - - -
Last change: 25/08/95-07:48

3.2.3 caiRWTable, caiReadTable, caiWriteTable(3)

NAME

caiRWTable,caiReadTable,caiWriteTable -
Direct R/W table record(s)/field(s) in database

SYNOPSIS

```
#include "cai.h"

ccsCOMPL_STAT caiRWTable (
                           IN  char       *table,
                           IN  vltLOGICAL  write,
                           IN  int attrNum,  IN void *attrBuffer,
                           IN  int firstRec, IN int   lastRec,
                           IN  int firstFld, IN int   lastFld,
                           OUT ccsERROR   *error
                          )
ccsCOMPL_STAT caiReadTable, caiWriteTable (
                           IN  char       *table,
                           IN  int attrNum,  IN void *attrBuffer,
                           IN  int firstRec, IN int   lastRec,
                           IN  int firstFld, IN int   lastFld,
                           OUT ccsERROR   *error
                          )
```

DESCRIPTION

This routine read/write the attribute indexed by <attrNum> from/to the database. The addressing is direct, the address is read out of the table <table>. The attribute's value is pointed to by <attrBuffer>. The flag <write> selects the operation to perform.

```
table      - pointer to attribute table
write      - operation type (FALSE : read ; TRUE : write)
attrNum    - attribute's index in attribute table
attrBuffer - pointer to attribute's memory space
firstRec,lastRec - first/last table contiguous records to access
firstFld,lastFld - first/last table record contiguous fields to access
error      - error handling data structure
```

If anything goes wrong FAILURE is returned and the error reason and description is returned in the <error> variable.

RETURN VALUES

SUCCESS if everything ok
FAILURE if anything went wrong

CAUTION

It is the responsibility of the designer to provide enough memory space for reading the database attributes.

- - - - - -
Last change: 25/08/95-07:48

3.2.4 caiRWVector, caiReadVector, caiWriteVector(3)

NAME

caiRWVector,caiReadVector,caiWriteVector -
Direct R/W vector element(s) in database

SYNOPSIS

```
#include "cai.h"

ccsCOMPL_STAT caiRWVector (
                            IN  char       *table,
                            IN  vltLOGICAL  write,
                            IN  int attrNum,  IN void *attrBuffer,
                            IN  int firstElt, IN int   lastElt,
                            OUT ccsERROR   *error
                           )
ccsCOMPL_STAT caiReadVector, caiWriteVector (
                            IN  char       *table,
                            IN  int attrNum,  IN void *attrBuffer,
                            IN  int firstElt, IN int   lastElt,
                            OUT ccsERROR   *error
                           )
```

DESCRIPTION

This routine read/write the attribute indexed by <attrNum> from/to the database. The addressing is direct, the address is read out of the table <table>. The attribute's value is pointed to by <attrBuffer>. The flag <write> selects the operation to perform.

```
table      - pointer to attribute table
write      - operation type (FALSE : read ; TRUE : write)
attrNum    - attribute's index in attribute table
attrBuffer - pointer to attribute's memory space
firstElt,lastElt - first/last vector contiguous elements to access
error      - error handling data structure
```

If anything goes wrong FAILURE is returned and the error reason and description is returned in the <error> variable.

RETURN VALUES

SUCCESS if everything ok
FAILURE if anything went wrong

CAUTION

It is the responsibility of the designer to provide enough memory space for reading the database attributes.

```
- - - - - -
Last change:  25/08/95-07:48
```

3.2.5 caiGetAttrInfo(3)

NAME

caiGetAttrInfo - get the attribute's information

SYNOPSIS

```
#include "cai.h"

void caiGetAttrInfo (
                      IN  char       *table,
                      IN  int         attrNum,
                      OUT dbATTRTYPE *attrTyp,
                      OUT vltUINT8   *fldCnt,
                      OUT vltUINT16  *recCnt,
                      OUT vltUINT32  *recSize,
                      OUT vltUINT16  *recsUsed,
                      OUT dbTYPE     *dtaTyp,
                      OUT ccsERROR   *error
                     )
```

DESCRIPTION

This routine returns the attribute's information.

```
table     - address of attribute table
attrNum   - attribute's index in attribute table
attrTyp   - pointer to attribute type (scalar/table/vector)
fldCnt    - pointer to number of fields  (table)
recCnt    - pointer to number of records
recSize   - pointer to record size
recsUsed  - pointer to number of used records
dtaTyp    - pointer to array of data types
error     - error handling data structure

The parameters <attrTyp>, <fldCnt>, <recCnt>, <recSize>, <recsUsed>
and <dtaTyp> may be defaulted to NULL, if they are not relevant.

If <dtaTyp> is not NULL, it MUST contain at least 2 elements.

If anything goes wrong FAILURE is returned and the error reason and
description is returned in the <error> variable.
```

RETURN VALUES

```
SUCCESS if everything ok
FAILURE if anything went wrong
```

CAUTIONS

```
No check is made on the vector <dtaTyp>. It is assumed that its
length is at least the expected attribute <fldCnt> + 1. For security,
it is recommended to allocate dbMAX_FIELD_CNT + 1 elements.
The parameter <recsUsed> is NOT updated (preset to <recCnt>.

- - - - - -
Last change:  27/09/95-07:48
```

3.2.6 caiFreeAttrTable(3)

NAME

caiFreeAttrTable - free memory space of attributes table

SYNOPSIS

```
#include "cai.h"

void caiFreeAttrTable (
                        IN  char *table
                       )
```

DESCRIPTION

This routine returns to the system all the memory space pointed to by <table>.

- - - - - -
Last change: 21/06/95-07:48

3.3 Miscellaneous

3.3.1 caiVersion(1)

NAME

caiVersion - Return/Display Common Access Interface Version

SYNOPSIS

```
#include "cai.h"

void caiVersion ( vltBYTES80 version )
```

DESCRIPTION

Extracts the version number of the installed Common Access Interface. If <version> is NULL, the line is printed to the console.

RETURN VALUES

none

- - - - - -
Last change: 21/06/95-07:48

3.4 Include File

```
/*****************************************************************************
* cai.h - common access interface functions interface file
*
* This header file contains data type definitions and function prototypes
* for the resolution and loading of SDL function entry and attribute
* direct address tables.
*----------------------------------------------------------------------------
*/
#ifndef CAI_H
#define CAI_H
/*****************************************************************************
 * Header Files
 ****************************************************************************/
#ifdef __cplusplus
extern “C” {
#endif

#include <stdio.h>

#include “CCS.h”                          /* CCS/LCC common definitions */
#include “caiErrors.h”                    /* CAI error codes            */

/* literal for CAI library version */
#define caiVERSION      “$Revision$”
#define caiVERSION_DATE “SEP95”

/* CAI module name */
#define caiMODULE       “cai”
#define caiMODULE_NAME  “Common Access Interface”

/* Generic SDL function prototype */
typedef ccsCOMPL_STAT (* caiVFUNCTION)( void *desc, ccsERROR *error,
                                        va_list parList );
/* Function prototypes */
#ifdef MAKE_VXWORKS
ccsCOMPL_STAT caiGetFunctions  ( char **table, vltBYTES8 prefix,
                                 char *funcNames[], ccsERROR *error );
ccsCOMPL_STAT caiCallFunction  ( char *table, int caiNum,
                                 ccsERROR *error, void *desc, ... );
void          caiFreeFuncTable ( char *table );
#endif

ccsCOMPL_STAT caiGetAttributes ( char **table, dbSYMADDRESS point,
                                 char *attrNames[], ccsERROR *error);
ccsCOMPL_STAT caiGetAttrInfo   ( char *table, int attrNum,
                                 dbATTRTYPE *attrTyp, vltUINT8 *fldCnt
                                 vltUINT16 *recCnt, vltUINT32 *recSize,
                                 vltUINT16 *recsUsed, dbTYPE *dtaTyp,
                                 ccsERROR *error);
ccsCOMPL_STAT caiRWAttribute   ( char *table, vltLOGICAL write,
                                 int attrNum, void *attrBuffer,
                                 ccsERROR *error );
ccsCOMPL_STAT caiReadAttribute ( char *table, int attrNum,
                                 void *attrBuffer, ccsERROR *error);
ccsCOMPL_STAT caiWriteAttribute( char *table, int attrNum,
                                 void *attrBuffer, ccsERROR *error);
```

```
ccsCOMPL_STAT caiRWTable       ( char *table, vltLOGICAL write,
                                 int attrNum, void *attrBuffer,
                                 int firstRec, int lastRec,
                                 int firstFld, int lastFld,
                                 ccsERROR *error);
ccsCOMPL_STAT caiReadTable     ( char *table,
                                 int attrNum, void *attrBuffer,
                                 int firstRec, int lastRec,
                                 int firstFld, int lastFld,
                                 ccsERROR *error);
ccsCOMPL_STAT caiWriteTable    ( char *table,
                                 int attrNum, void *attrBuffer,
                                 int firstRec, int lastRec,
                                 int firstFld, int lastFld,
                                 ccsERROR *error);
ccsCOMPL_STAT caiRWVector      ( char *table, vltLOGICAL write,
                                 int attrNum, void *attrBuffer,
                                 int firstElt, int lastElt,
                                 ccsERROR *error);
ccsCOMPL_STAT caiReadVector    ( char *table,
                                 int attrNum, void *attrBuffer,
                                 int firstElt, int lastElt,
                                 ccsERROR *error);
ccsCOMPL_STAT caiWriteVector   ( char *table,
                                 int attrNum, void *attrBuffer,
                                 int firstElt, int lastElt,
                                 ccsERROR *error);
void          caiFreeAttrTable ( char *table );

void          caiVersion       ( vltBYTES80 version );

#ifdef __cplusplus
}
#endif

#endif /* if !CAI_H */

/*****************************************************************************/

/*___oOo___*/
```

4 INSTALLATION GUIDE

This chapter describes the procedures to perform in order to build the **cai** module and **libcai.a** library and how to install the **cai** module on the target LCU.

4.1 Installation requirements

4.1.1 Hardware Requirements

Required hardware as defined in section 1.2.
Required disk space approx. 100 kBytes.
The **cai** module occupies approx. 5 kBytes on the LCU.
The **libcai.a** library occupies approx. 190kB on the WS.

4.1.2 Software Requirements

Required software as defined in section 1.2.

4.2 Building the Software

4.2.1 Delivery

The delivery considerations of [5] apply here.
The Common Access Software is delivered in directories as defined in [1].

```
./cai/
     |--ws/
     |     |--src/
     |          |--Makefile
     |     |--include/
     |          |--cai.h
     |          |--caiInternal.h
     |          |--caiErrors.h
     |     |--ERRORS/
     |          |--cai_ERRORS
     |--lcu/
     |     |--dbl/
     |          |--caiTest.db
     |     |--src/
     |          |--Makefile
     |          |--caiGetFunctions.c
     |          |--caiCallFunction.c
     |          |--caiFreeFuncTable.c
     |          |--caiGetAttributes.c
     |          |--caiGetAttrInfo.c
     |          |--caiRWAttribute.c
     |          |--caiRWTable.c
     |          |--caiRWVector.c
     |          |--caiFreeAttrTable.c
     |          |--caiVersion.c
```

4.2.2 Build Procedure for LCU

If the module **cai** is to be built, a makefile is provided in the **`./lcu/src`** sub-directory.

Before (re-)compiling the software, it is recommended to clean the delivered environment, using the **clean** target.

```
% make clean
```

The following command (re)compiles and links the LCU Common Access Interface software.

```
& make all
```

Before installing the software, the manual pages must also be (re-)built, using the **man** target.

```
% make man
```

The **install** target will execute the necessary file transfers to the environment **`$INTROOT`** or **`$VLTROOT`**:

```
% make install
```

4.2.3 Build Procedure for WS

If the library **libcai.a** is to be built, a makefile is provided in the **`./ws/src`** sub-directory.

Before (re-)compiling the software, it is recommended to clean the delivered environment, using the **clean** target.

```
% make clean
```

The following command (re)compiles and archives the Common Access Interface software.

```
& make all
```

Before installing the software, the manual pages must also be (re-)built, using the **man** target.

```
% make man
```

The **install** target will execute the necessary file transfers to the environment **`$INTROOT`** or **`$VLTROOT`**:

```
% make install
```

4.3 VxWorks Environment Configuration

The boot script delivered with the LCU Common Software shall be modified, so that the **cai** module be loaded on the target LCU.

After LCC has been loaded (section 4 of the boot script), add the following line:

```
lcu-> ld < cai
```

In order to verify the installation, call the **caiVersion** function directly from the LCU shell:

```
lcu-> caiVersion
```

which shall display

```
cai - Common Access Interface - Version 1.14 - SEP95
```

5 ERRORS MESSAGES

For error handling, the CCS/LCC standard error mechanism is used. The error structure must be provided by the designer and passed by reference to the **cai** functions.

For detailed information about the LCC standard error handling mechanism, see [3].

Possible errors logged by this module are listed below, as defined in **caiErrors.h**.

- caiERR_PARAMETER 1 S invalid parameter %s
- caiERR_ENTRY 2 S no entry to function %s
- caiERR_CALL_FUNC 3 S call to function failed
- caiERR_SYMBOL 4 S entry %s not found in global symbol table
- caiERR_ATTRIBUTE 5 S invalid attribute %s
- caiERR_POINT 6 S invalid point %s
- caiERR_READ 7 S database direct read attribute failed
- caiERR_WRITE 8 S database direct write attribute failed
- caiERR_MEMORY 9 S cannot allocate memory space for entry table

___ oOo ___