OOWG Tutorial

Programming with eccs

G.Chiozzi

22/02/96

Contents

- **Introduction**
- **Message Handling**
- **Database Handling**
- **Error Handling**
- **Conclusion**

INTRODUCTION

Extended CCS is a library written in C++ on top of CCS. It contains functions and simple classes and requires a minimal knowledge of C++ syntax.

It runs only on Workstation.

Purpose of the library is:

- Provide easier usage of common CCS features
- Embed in functions or simple classes commonly used application patterns
- Provide base CCS classes for the development of truly object oriented applications
- Prevent common errors in applications
- Standardise produced code

The library is not (and does not want to be) a complete C++ replacement of CCS.
Only the commonly used parts of CCS are integrated in ECCS.

ECCS Features

- General features (macros, defines, basic functions)
- Message system
- Database access
- Error handling
- Database vent handling
- Logging system
- Fast prototyping functions

Using ECCS

In the Makefile:

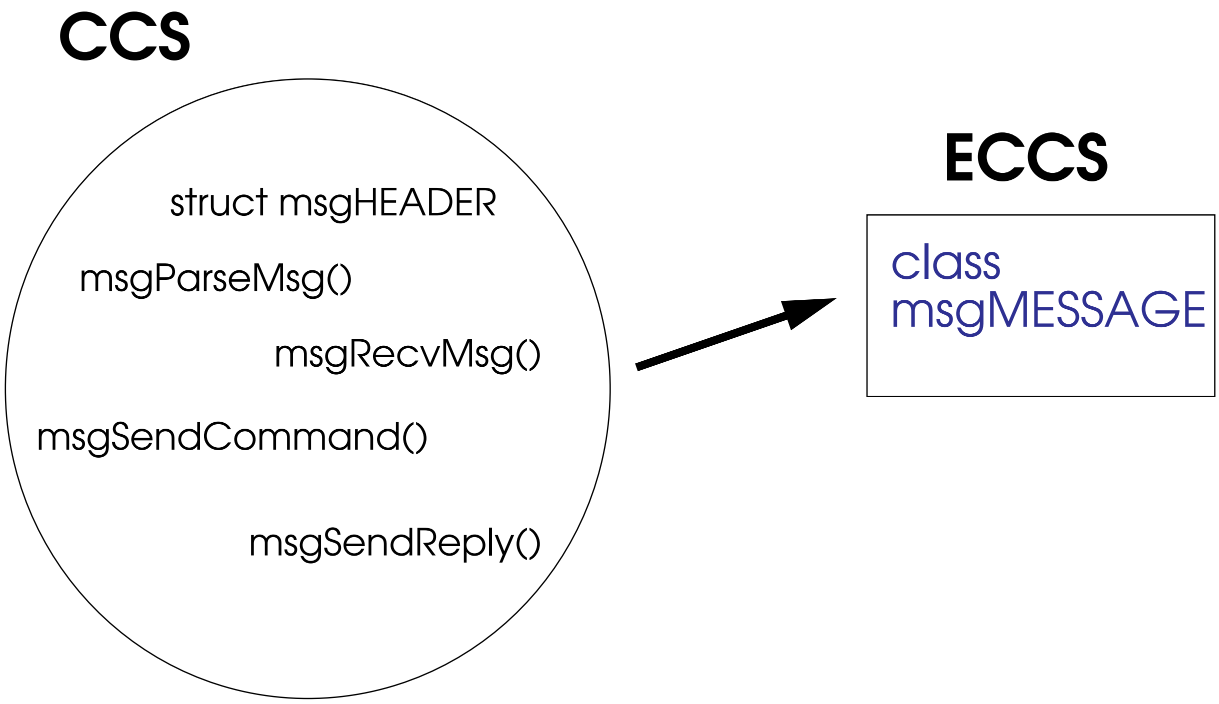
- link the CCS standard libraries
- link the C++ standard libraries (`stdc++, g++, iostream, iberty`)
- link the required support library `fnd`
- link the ECCS library `eccs`

```
Example:

EXECUTABLE=eccsTest
TESTLIBS = eccs fnd stdc++ g++ iostream iberty

eccsTest_OBJECTS= eccsTest
eccsTest_LIBS   = ${TESTLIBS} CCS
```

In the code:

```
#include “ECCS.h”
int main(int, char *argv[])
{
    ccsCOMPL_STAT stat;
    stat = ccsInit(argv[0]);
    eccsErrLog(stat,”Error in Init “);
    ..... application code .....
    stat = ccsExit();
    eccsErrLog(stat,”Error in ccsExit”);
}
```

Message Handling



Example: sending a command

CCS sample code

```
 #include "msg.h"

 ...
ccsERROR        error;
msgCMD          command;
ccsENVNAME      destEnv;
ccsPROCNAME     destProc;
char            cmdpar[] = "my cmd buffer";
...

...
strcpy((char *)destProc, argv[1]);
strcpy((char *)destEnv, argv[2]);
strcpy((char *)command,argv[3]);

 if (msgSendCommand (command, 0, destEnv, destProc,
                    cmdpar,strlen(cmdpar) +1,
                    0, &error) == FAILURE)
    ... error processing ...
else
    ... normal processing ...
```

eccs sample code

```
 #include "msgMESSAGE.h"

 ...

 msgMESSAGE msg;

 msg.Procname(argv[1])
     .Destenv(argv[2])
     .Command(argv[3])
     .Buffer("my cmd buffer");

 if (msg.SendCommand() == FAILURE)
    ... error processing ...
else
    ... normal processing ...
```

msgMESSAGE class (public interface)

```
class  msgMESSAGE: public msgRAW_MESSAGE {
  public:
    msgMESSAGE();
    msgMESSAGE(const msgHEADER *msg);
    msgMESSAGE(const msgMESSAGE &source);

    ccsCOMPL_STAT Parse(const msgHEADER *msg);
    ccsCOMPL_STAT Receive(msgTIMEOUT timeout = msgNO_TIMEOUT,
                const msgRECEIVEFILTER *filter  = NULL);

    ccsCOMPL_STAT SendCommand();
    ccsCOMPL_STAT SendCommand(msgCHECKFLAG flag);

    ccsCOMPL_STAT SendReply();
    ccsCOMPL_STAT SendReply(const char *newBuffer,
                msgLENGTH   newBuflen=0,
                vltLOGICAL  lastReply=ccsTRUE);
    ccsCOMPL_STAT SendReply(const ccsERROR &errmsg);


    /* Methods to access safely the values of parameters */
    vltUINT8             Type() const;
    msgMESSAGE          &Type(vltUINT8 newType);

    const char          *Command() const;
    msgMESSAGE          &Command(const msgCMD newCommand);

    msgCMDID             CommandId() const;
    msgMESSAGE          &CommandId(msgCMDID newCommandId);

    const char          *Destenv() const;
    msgMESSAGE          &Destenv(const ccsENVNAME newDestenv);

    const char          *Destproc() const;
    msgMESSAGE          &Destproc(const ccsPROCNAME newDestProc);
```

```
    const msgPROCESSID &OrgId() const;
    msgMESSAGE         &OrgId(const  msgPROCESSID &newOrgId);

    vltLOGICAL          LastReply() const;
    msgMESSAGE         &LastReply(vltLOGICAL newLastReply);

    const char         *Buffer() const;
    msgLENGTH           Buflen() const;
    msgMESSAGE         &Buffer(const char *newBuffer,
                                       msgLENGTH newBuflen=0);

    ccsERROR           &Errmsg();
    msgMESSAGE         &Errmsg(const ccsERROR &newErrmsg);

    msgCHECKFLAG        Check() const;
    msgMESSAGE         &Check(msgCHECKFLAG flag);

    msgMESSAGE         &operator =(const msgMESSAGE &source);

    static  ccsCOMPL_STAT GlobalCheckOn(msgCHECKFLAG flag);
    static  ccsCOMPL_STAT GlobalCheckOff();

}
```

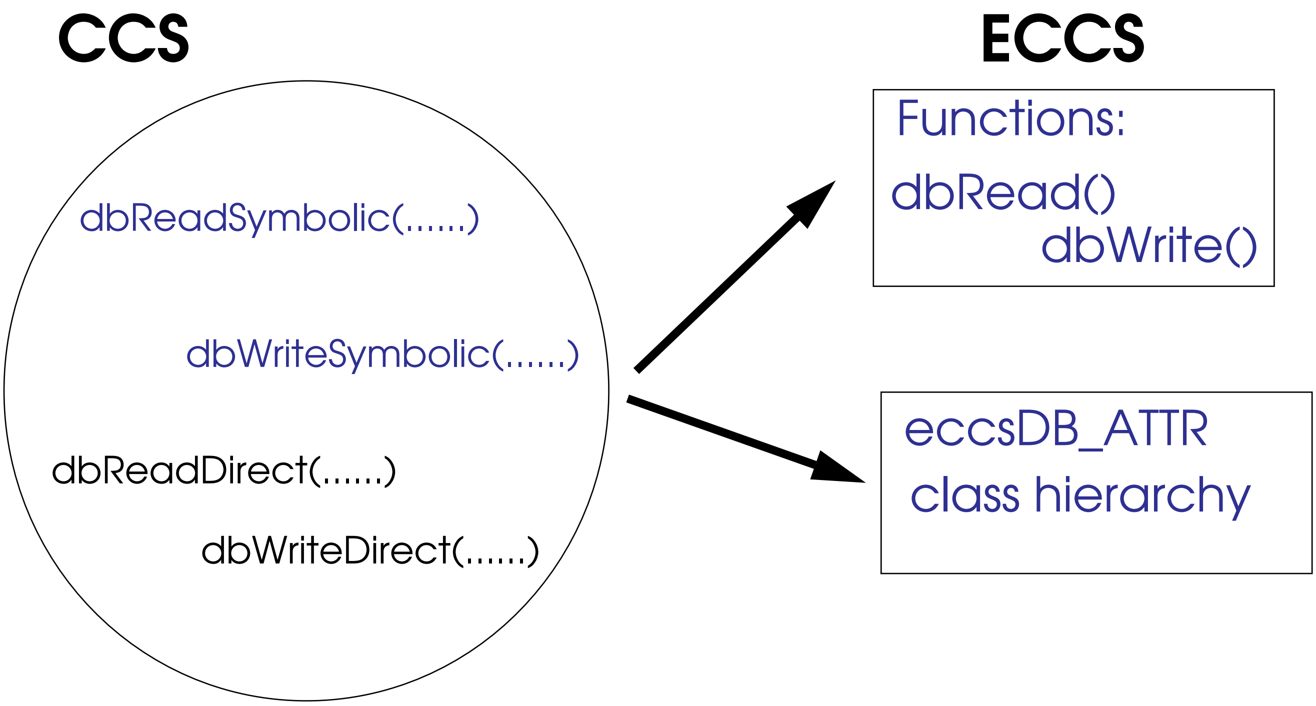
Example:
receive a command -> send reply

```
// Declare the message variable
msgMESSAGE myMessage;

// Receive the command (by default with no timeout and no filtering)
myMessage.Receive();

// Check for the command
if( strcmp(myMessage.Command(), "MYCMD) != 0) return;

// Send the last reply back to the sender of the message
// setting at the same time the message buffer to be send
myMessage.SendReply("This is my reply");
```

Example: send a command -> receive reply

```
// Declare the message variable
msgMESSAGE myMessage;

// Set the command parameters
myMessage.Command("MYCMD");
myMessage.Destenv("wte16");
myMessage.Procname("MYPROC");
myMessage.Buffer("Data sent");

// Send the command
myMessage.SendCommand();

// Receive the reply
int flag = ccsFALSE;
while(flag == ccsFALSE)
    {
    // (checking of the specific reply is not shown)
    myMessage.Receive();

    flag = myMessage.LastReply();

    printf("Reply buffer: %s\n", myMessage.Buffer());
    }
```

Database Handling



Example: sending a command

CCS sample code

```
int main(int argc, char *argv[])
{
    dbSYMADDRESS  pointName;
    dbATTRIBUTE   attrName;
    dbTYPE        dataType[dbMAX_FIELD_CNT];

    vltINT32   value;

    vltINT32      actual;
    vltUINT16     recordCnt;
    dbATTRTYPE    attrType;
    ccsERROR      error;

    ccsInit(argv[0],0,NULL,NULL,&error);
    dbOpen(ccsLOCAL_ENV,&error);

    strcpy((char *) pointName,":PARAMS:SCALARS");
    strcpy((char *) attrName,"scalar_int32");

    if ( dbReadSymbolic(pointName,attrName,dataType,(char*)&value,
         sizeof(value), &actual,&recordCnt,&attrType,&error)== FAILURE ){}
    value = value + 5;
    if ( dbWriteSymbolic(pointName,attrName,dataType, (char*)&value,
         sizeof(value), &actual,recordCnt,attrType,&error) == FAILURE ){}

    if ( dbReadSymbolic(pointName,attrName,dataType, (char*)&value,
         sizeof(value), &actual,&recordCnt,&attrType,&error)== FAILURE ){}

    ccsExit(&error);
    return 0;
}
```

eccs sample code

Using dbRead() and dbWrite():

```
int main(int, char *argv[])
{
    ccsInit(argv[0]);
    dbOpen();

    vltINT32 value;

    if( dbRead(&value,":PARAMS:SCALARS.scalar_int32") ) {}

    value = value + 5;
    if( dbWrite(&value,":PARAMS:SCALARS.scalar_int32") ) {}

    if( dbRead(&value,":PARAMS:SCALARS.scalar_int32") ) {}

    ccsExit();
    return 0;
}
```

Using eccsDB_ATTR classes:

```
int main(int, char *argv[])
{
    ccsInit(argv[0]);
    dbOpen();

    eccsDB_INT32 value(":PARAMS:SCALARS.scalar_int32");

    value = (vltINT32)value + 5;
    ccsExit();
    return 0;
}
```

eccsDB_ATTR classes

Basic data types:

- eccsDB_ATTR
- eccsDB_LOGICAL
- eccsDB_INT16
- eccsDB_INT32
- eccsDB_FLOAT
- eccsDB_DOUBLE
- eccsDB_STRING

Database vectors:

- eccsDB_VECTOR_vltINT8
- eccsDB_VECTOR_vltUINT8
- eccsDB_VECTOR_vltINT16
- eccsDB_VECTOR_vltUINT16
- eccsDB_VECTOR_vltINT32
- eccsDB_VECTOR_vltUINT32
- eccsDB_VECTOR_vltLOGICAL
- eccsDB_VECTOR_vltDOUBLE
- eccsDB_VECTOR_vltFLOAT
- eccsDB_VECTOR_vltBYTESxx

Database tables:

Tables are implemented using a template class since it is not possible to provide pre-defined classes:

- eccsDB_TABLE<MY_TABLE_RECORD>

The record definition class MY_TABLE_RECORD must be a subclass of eccsDB_TABLE_RECORD

Example: database vectors

```
#include "eccsDB_VECTOR.h"

int main(int, char *argv[])
{
    ccsCOMPL_STAT stat,st;

    stat = ccsInit(argv[0]);
    stat = dbOpen();

    vltINT32 recordCnt;
    vltINT32 i;

    eccsDB_VECTOR_vltINT32 vector(":PARAMS:VECTORS.vector_int32");

    recordCnt = vector.RecordCnt();

    vltINT32 *array = new vltINT32[recordCnt];

    st = vector.GetValue(array);

    st = vector.SetValue(array);

    // Testing subscript operator
    for(i =0; i<recordCnt; i++) {
    vector[i] = recordCnt-i;
    cout << " " << (vltINT32)vector[i];
    }

    for(i =0; i<recordCnt; i++) {
    array[i] = vector[i];
    cout << " " << array[i];
    }

    stat = dbExit();
}
```

Example: database tables

```
#include "eccsDB_TABLE.h"

// user provided TABLE class for db table TABLES.full_table
class TABLE : public eccsDB_TABLE_RECORD
{
  public:
    vltLOGICAL logical;
    vltINT8    int8;
    vltUINT32  uint32;
    ccsCOMPL_STAT Unpack( void *buffer )    {
        memcpy( &logical, buffer,sizeof(vltLOGICAL));
        buffer += sizeof(vltLOGICAL);
        memcpy( &int8,    buffer,sizeof(vltINT8)   );
        buffer += sizeof(vltINT8);
        memcpy( &uint32,  buffer,sizeof(vltUINT32) );

        return(SUCCESS);
    }
    ccsCOMPL_STAT Pack( void *buffer )    {
        int size;
        dbFillBuf( (char**)&buffer, (char*)&logical  ,&size, dbLOGICAL);
        dbFillBuf( (char**)&buffer, (char*)&int8     ,&size, dbINT8);
        dbFillBuf( (char**)&buffer, (char*)&uint32   ,&size, dbUINT32);
        return(SUCCESS);
    }

};


int main(int, char *argv[])
{
    ccsCOMPL_STAT stat,st;

    stat = ccsInit(argv[0]);
    stat = dbOpen();
```

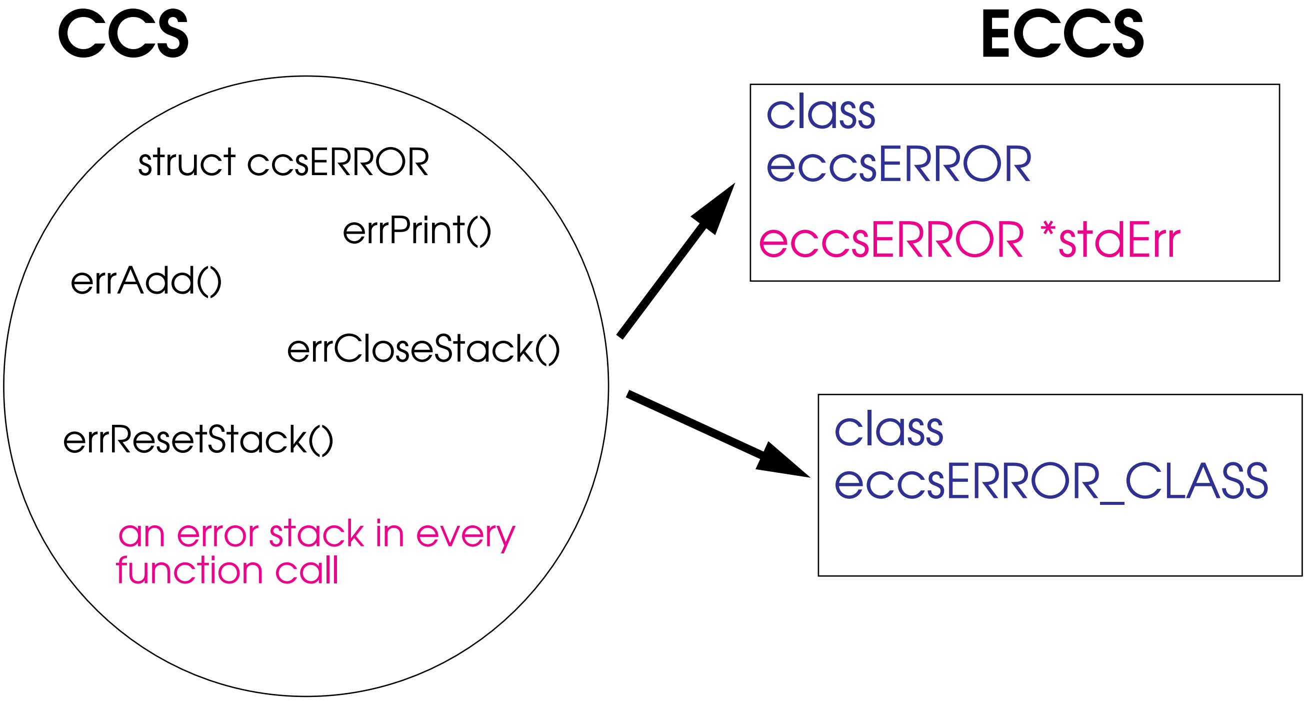
```
    vltINT32 recordCnt;

    // Define a TABLE
    eccsDB_TABLE<TABLE> table("<alias>TABLES.full_table");
    if ( table.ErrStatus() == FAILURE ){}

    // Get the number of records in the array
    recordCnt = table.RecordCnt();

    // Allocate the memory array
    TABLE *table_array = new TABLE[recordCnt];

    //Read all table
    st = table.GetValue(table_array);

    // Modify values in memory
    for ( i=0;i<recordCnt;i++)
    {
    table_array[i].logical=TRUE;
    table_array[i].int8     = i;
    table_array[i].uint32   = i;
    }


    // Rewrite whole table with changes
    st = table.SetValue(table_array);

    // Read one by one
    table_array[i] = table[i];

    stat = dbExit();

} /* end main() */
```

Example: database tables 2

```
#include "eccsDB_TABLE.h"

class simREPLY_TABLE : public eccsDB_TABLE_RECORD
{
  public:
    vltBYTES8   command;
    vltBYTES256 reply;
    ccsCOMPL_STAT Unpack( void *buffer )    {
        memcpy( &command, buffer,sizeof(command) );
        buffer = buffer+sizeof(command);
        memcpy( &reply, buffer,sizeof(reply));
        return(SUCCESS);
    }
    ccsCOMPL_STAT Pack( void *buffer )    {
        int size;
        dbFillBuf( (char**)&buffer, (char*)&command ,&size, dbBYTES8);
        dbFillBuf( (char**)&buffer, (char*)&reply  ,&size, dbBYTES256);
        return(SUCCESS);
    }

    void Print()    {
        cout << "Command: " << command << "  - Reply:  " << reply;
        cout << endl;
    }
};

int main(int, char *argv[])
{
    /************/
    /* Init     */
    /************/

    ccsCOMPL_STAT stat,st;

    stat = ccsInit(argv[0]);
```

```
    stat = dbOpen();

    vltINT32 recordCnt;

    // Define a TABLE
    simREPLY_TABLE testRec;

    eccsDB_TABLE<simREPLY_TABLE> table("<alias>eccsTest:tab.replies");
    if ( table.ErrStatus() == FAILURE ){}

    // Get the number of records in the array
    recordCnt = table.RecordCnt();

    // Allocate the memory array
    simREPLY_TABLE *table_array = new simREPLY_TABLE[recordCnt];

    /* Read random values and print contents */
    cout << "Read values by contents: " << endl;

    testRec = table["COMMENT"]; testRec.Print();
    testRec = table["SETUP"]; testRec.Print();

    stat = dbExit();
}

// __oOo__
```

Error Handling



The class eccsERROR

The implementation of the eccsERROR class provides the following advantages:

- Any time an instance of eccsERROR is deleted (or exit out of scope), it is automatically checked and the stack is closed if it contains errors.
- The defaults error stack is automatically closed when the application exits.

```
struct eccsERROR: public ccsERROR
{
  eccsERROR();
 ~eccsERROR();

  ccsCOMPL_STAT  Add(ccsMODULEID moduleId, vltINT16 errorNumber,
                 ccsLOC_ID locId, ...);
  ccsCOMPL_STAT  Add_v(ccsMODULEID moduleId, vltINT16 errorNumber,
                 ccsLOC_ID locId, va_list va_parList);
  ccsCOMPL_STAT  SysAdd(ccsMODULEID moduleId, vltINT16 errorNumber,
                 ccsLOC_ID locId, char *sysModule=modRTAP, ...);
  ccsCOMPL_STAT  SysAdd_v(ccsMODULEID moduleId, vltINT16 errorNumber,
                 ccsLOC_ID locId, char *sysModule,
                 va_list va_parList);

  ccsCOMPL_STAT  Close();
  ccsCOMPL_STAT  Display(vltLOGICAL syncFlag=FALSE);
  ccsCOMPL_STAT  Print();
  ccsCOMPL_STAT  Reset();

  eccsERROR &operator =(const ccsERROR &e);

  /* This static section is used to handle default eccs error stack */
  static void DefErrorStack(eccsERROR *err = NULL);
};
```

Examples:

```
// ========================================================
// Calling an ECCS function and handling errors with the
// default error stack
ccsCOMPL_STAT status;

status = ccsInit(argv[0]);
if(status == FAILURE)
{
    // Notice that the stack is closed automatically on exit
    // and I do not need to close it explicitly
    stdErr->Add(....parms....);
    // This line is equivalent to:
    // errAdd(...params....);    // uses stdErr by default
    exit(1);
}

// ============================================================
// Calling a standard CCS function and handling errors with the
// default error stack
ccsCOMPL_STAT status;

status = dbMultiRead(listid,stdErr);
if(status == FAILURE)
{
    stdErr->Add(....parms....);
}
// ============================================================
```

The class eccsERROR_CLASS

All the classes that want to provide error handling must be derived from the class eccsERROR_CLASS.

```
class eccsERROR_CLASS
{
  public:

    eccsERROR_CLASS(eccsERROR *s = stdErr);

    ccsCOMPL_STAT  ErrStatus();
    eccsERROR     &ErrStack();

    ccsCOMPL_STAT  ErrStackClose();
    ccsCOMPL_STAT  ErrStackReset();

    ccsCOMPL_STAT  ObjStatus();

    operator ccsCOMPL_STAT();
    operator eccsERROR &();

  protected:
    eccsERROR_CLASS &ErrStatus(ccsCOMPL_STAT s);
    eccsERROR_CLASS &ErrReset();
    eccsERROR_CLASS &ObjStatus(ccsCOMPL_STAT s);

    eccsERROR_CLASS &operator =(const ccsCOMPL_STAT s);

    ccsCOMPL_STAT  ErrAdd(ccsMODULEID moduleId, vltINT16 errorNumber,
              ccsLOC_ID locId, ...);
    ccsCOMPL_STAT  ErrAdd_v(ccsMODULEID moduleId, vltINT16 errorNumber,
```

```
             ccsLOC_ID locId, va_list va_parList);
    ccsCOMPL_STAT  ErrSysAdd(ccsMODULEID moduleId, vltINT16 errorNumber,
              ccsLOC_ID locId, char *sysModule=modRTAP, ...);
    ccsCOMPL_STAT  ErrSysAdd_v(ccsMODULEID moduleId, vltINT16 errorNumber,
                ccsLOC_ID locId, char *sysModule,
                va_list va_parList);

};
```

Example: defining a new class with error handling

```
class eccsDB_ATTR : public eccsERROR_CLASS
{
  public:
    eccsDB_ATTR(const dbSYMADDRESS pointName);
    eccsDB_ATTR(const eccsDB_ATTR &attr);
    virtual ~eccsDB_ATTR();

    virtual ccsCOMPL_STAT GetValue(void *buffer);
    virtual ccsCOMPL_STAT SetValue(void *buffer);
    virtual ccsCOMPL_STAT DbRead(void *buffer);

};


ccsCOMPL_STAT eccsDB_ATTR::GetValue(void *buffer)
{
    ccsCOMPL_STAT stat = SUCCESS;

    // Always call ErrReset() first, except in constructor, destructor
    // and special cases with nested functions
    ErrReset();

    stat = DbRead(buffer);
    if(stat == FAILURE)
    {
        ErrAdd("eccs",eccsERR_READATTR,__FILE_LINE__,symAddr,dbEMPTY);
        return stat;
    }

    return stat;

} /* eccsDB_ATTR::GetValue() */
```

Using error handling features:

```
eccsDB_ATTR attribute("<alias>TABLES.full_table");
if ( attribute.ErrStatus() == FAILURE )
{
cout << "failure Invalid table" << endl;
exit(1);
}


attribute.GetValue(buffer);
if ( attribute.ErrStatus() == FAILURE )
{
attribute.ErrStack().Add(....)
}

attribute.GetValue(buffer);
if ( attribute.ErrStatus() == FAILURE )
{
// The error is recovered
attribute.ErrStackReset();
}


// Store somewhere an error stack (for example to send
// an error reply)
eccsERROR errStack = attribute.ErrStack();
```

Conclusion

This presentation has shown only some of the most important aspects of the library.

The bigger benefits come using this library in congiunction with the evh library

For more details look at:

- VLT Software Extended CCS User Manual (VLT-MAN-ESO-17210-0770)
- Code examples in eccs <mod>/test directory