OOWG Tutorial
Programming with evh

G.Chiozzi

07/03/96

Contents

INTRODUCTION

The Event Handling Toolkit is a set of C++ classes to develop event driven applications on workstations.
Development of applications requires a knowledge of concepts for event driven programs and object-oriented programming techniques.

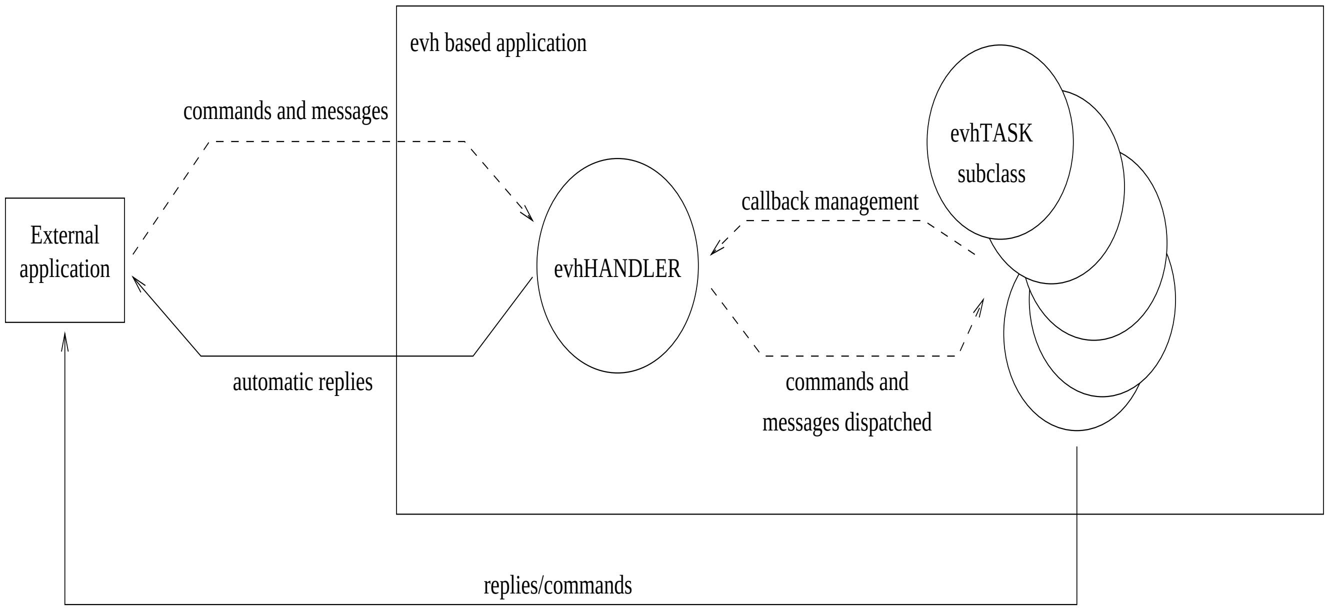
It runs only on Workstation.

Purpose of EVH is:

- Provide a framework to implement easily workstation applications that receive and send commands and messages.
- Implement in a standard way common application patterns
- Prevent common errors in applications
- Support object-oriented design and development
- Provide tools for testing and debugging of applications.

The main loop

- Processes are designed in an event driven way
- The implementation is based around the *evhHANDLER* class



The design of every process consists mainly in the design of the independent objects providing the methods to be attached as callbacks to the events

evh features

- The following events are handled:
    - commands
    - replies and error replies
    - time out notifications
    - cyclic timers
    - data base events
    - alarms
    - signals
    - file input
- callbacks are stored in a dynamic dictionary
- callbacks must return within a "command response time" of 1 second
- a wide set of ready-made classes implements standard behaviors and event handling protocols

Using EVH

In the Makefile:

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

```
Example:

EXECUTABLE=evhTest
TESTLIBS = evh eccs fnd stdc++ g++ iberty

evhTest_OBJECTS= eccsTest
evhTest_LIBS    = ${TESTLIBS} CCS
```

In the code:

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

    evhHandler->MainLoop();

    stat = ccsExit();
    eccsErrLog(stat,”Error in ccsExit”);
}
```

Example 1: The basic code structure

```
#define _POSIX_SOURCE 1
#include "vltPort.h"

// Include files for EasyCCS and Evh Toolkit
#include "ECCS.h"
#include "evhHANDLER.h"

// Include for the basic class
#include "evhTASK.h"

/*****************************************/
/* Definition for the APPLICATION class    */
/*****************************************/
class APPLICATION: public evhTASK
{
  public:
    APPLICATION();

    // Definition for the callback to be used receiving the
    // "SAMPLE" command
    evhCB_COMPL_STAT  SampleCB(msgMESSAGE &msg, void *udata);

}; /* end class APPLICATION */


/* Body of the constructor method.              */
/* It is called whenever an instance is created */
APPLICATION::APPLICATION()
{
    /* This variable is used to define the event:    */
    /* a message of type msgTYPE_COMMAND, with       */
    /* Command="SAMPLE", no regard for the CommandId */
    evhMSG_TYPE_KEY key(msgTYPE_COMMAND,"SAMPLE");

    /* This variable is used to define the callback:     */
```

```
    /* It is a method of this object, no user parameters */
    evhOBJ_CALLBACK cb(this,(evhCB_METHOD)SampleCB);

    /* Then the callback is attached to the event handler */
    evhHandler->AddCallback(key,cb);
}

evhCB_COMPL_STAT  APPLICATION::SampleCB(msgMESSAGE &msg, void *)
{
    printf("Received a SAMPLE command: %s\n", msg.Buffer());

    /* Send back the reply to the command */
    msg.SendReply("Command received!");

    /* The callback must remain installed for another command */
    return evhCB_NO_DELETE;
}



/***********************************/
/* main procedure                 */
/***********************************/

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

    /* Create the instance of APPLICATION */
    APPLICATION app; USE(app);

    /* enter the main loop */
    evhHandler->MainLoop();

    ccsExit();
}
```

The standard VLT application

A standard VLT application can be implemented based on the evhSTD_COMMAND_CLASS, that provides:

- Standard commands:
  - PING
  - INIT
  - STATE,STATUS
  - STANDBY,ONLINE,OFF
  - SELFTEST, TEST
  - SIMULAT, STOPSIM
  - VERBOSE
  - VERSION
  - STOP
  - EXIT, KILL, BREAK
- Standard database point:

```
CLASS evhSTD_COMMANDS
BEGIN
    ATTRIBUTE INT32 state evhSTATE_OFF
    ATTRIBUTE LOGICAL simulation
    ATTRIBUTE LOGICAL verbose
END
```

Example 2: standard application

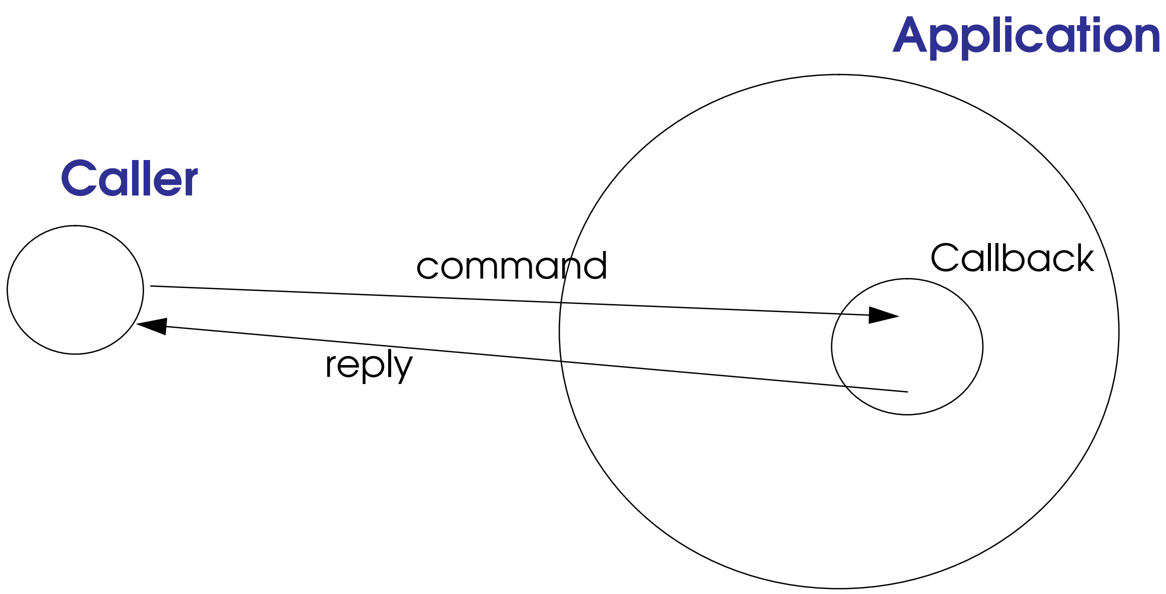
The following code (plus a proper point in the online database) defines a simple application fully compliant to VLT standards:

```
// Include files for EasyCCS and Evh Toolkit
#include "ECCS.h"
#include "evhHANDLER.h"

// Include for the basic class
#include "evhSTD_COMMANDS.h"


/**********************************/
/* main procedure                 */
/**********************************/

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

    /* Create the instance of APPLICATION */
    /* The call to the macro USE() prevents the GCC compiler     */
    /* from issuing a warning: it assumes that app is not used ! */
    evhSTD_COMMANDS app("Appl_data:evhTest"); USE(app);

    printf("Entering the main loop and waiting for commands!\n");

    /* enter the main loop */
    evhHandler->MainLoop();

    ccsExit();
}
```

Handling “immediate” commands

“Immediade” commands are the simples case: an application receive a command, deals with it and sends an immediate reply to the originator.

“Immediate” commands are implemented writing yust one callback to receive the command.



Example 3: an immediate command

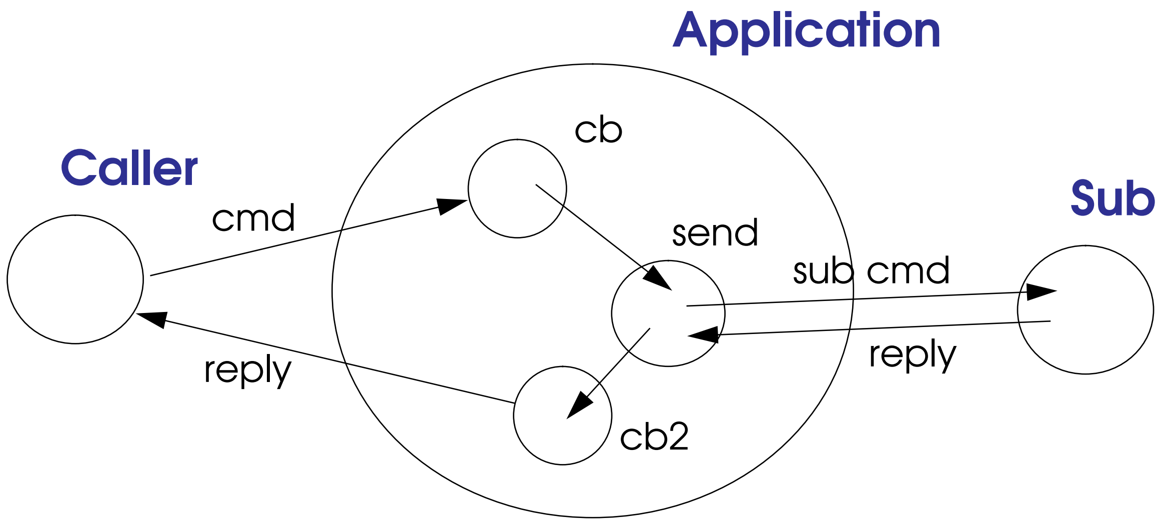
```
// Include files for EasyCCS and Evh Toolkit
#include "ECCS.h"
#include "evhHANDLER.h"

// Include for the basic class
#include "evhSTD_COMMANDS.h"

class APPLICATION: public evhSTD_COMMANDS
{
  public:
    APPLICATION(const dbSYMADDRESS  dbPoint);

    // Definition for the callback to be used receiving the
    // "SAMPLE" command
    evhCB_COMPL_STAT  SampleCB(msgMESSAGE &msg, void *udata);

}; /* end class APPLICATION */


/* Body of the constructor method.              */
/* It is called whenever an instance is created */
APPLICATION::APPLICATION(const dbSYMADDRESS  dbPoint) :
    evhSTD_COMMANDS(dbPoint)
{
    /* This variable is used to define the event:    */
    /* a message of type msgTYPE_COMMAND, with       */
    /* Command="SAMPLE", no regard for the CommandId */
    evhMSG_TYPE_KEY key(msgTYPE_COMMAND,"SAMPLE");

    /* This variable is used to define the callback:     */
    /* It is a method of this object, no user parameters */
    evhOBJ_CALLBACK cb(this,(evhCB_METHOD)SampleCB);

    /* Then the callback is attached to the event handler */
    evhHandler->AddCallback(key,cb);
```

```
}

/* Body for the callback */
evhCB_COMPL_STAT  APPLICATION::SampleCB(msgMESSAGE &msg, void *)
{
    /* Just print the message buffer on stdout */
    printf("Received a SAMPLE command: %s\n", msg.Buffer());

    /* Send back the reply to the command */
    msg.SendReply("Command received!");

    /* The callback must remain installed for another command */
    return evhCB_NO_DELETE;
}


/*********************************/
/* main procedure                */
/*********************************/

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

    /* Create the instance of APPLICATION */
    APPLICATION app("Appl_data:evhTest"); USE(app);

    printf("Entering the main loop and waiting for commands!\n");

    /* enter the main loop */
    evhHandler->MainLoop();

    ccsExit();
}
```

Handling “tranfer” commands

“Transfer” commands are a typical case for workstation coordination applications: an application receives a command, deals with it and send one sub-command to the controlled processes. When they are done it must sends the final reply to the originator.

“Transfer” commands are implemented using the evhCOMMAND or the evhDB_CMD_SEND class and some support callbacks to receive the command and send the replies.



Example 4: a transfer command

```
#define _POSIX_SOURCE 1
#include "vltPort.h"

#include "ECCS.h"
#include "evhHANDLER.h"
#include "evhCOMMAND.h"
#include "evhSTD_COMMANDS.h"

class APPLICATION: public evhSTD_COMMANDS
{
  public:
    APPLICATION(const dbSYMADDRESS  dbPoint);

    // Definition for the callback to be used receiving the
    // "SAMPLE" command
    evhCB_COMPL_STAT  SampleCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  EndSampleCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  ErrSampleCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  TimeoutSampleCB(msgMESSAGE &msg, void *udata);

  private:
    evhCOMMAND cmd;
    msgMESSAGE origMsg;

}; /* end class APPLICATION */


/* Body of the constructor method.              */
/* It is called whenever an instance is created */
APPLICATION::APPLICATION(const dbSYMADDRESS  dbPoint) :
    evhSTD_COMMANDS(dbPoint)
{
    evhMSG_TYPE_KEY key(msgTYPE_COMMAND);
    evhOBJ_CALLBACK cb(this,(evhCB_METHOD)SampleCB);
    ccsTIMEVAL      interval;
```

```
    /* Then the callback is attached to the event handler */
    evhHandler->AddCallback(key.Command("SAMPLE"),
                cb.Proc((evhCB_METHOD)SampleCB));

    interval.tv_sec  = 5;
    interval.tv_usec = 0;

    cmd.Reply(cb.Proc((evhCB_METHOD)EndSampleCB));
    cmd.ErrReply(cb.Proc((evhCB_METHOD)ErrSampleCB));
    cmd.Timeout(cb.Proc((evhCB_METHOD)TimeoutSampleCB),interval);
}

evhCB_COMPL_STAT  APPLICATION::SampleCB(msgMESSAGE &msg, void *)
{
    /* Just print the message buffer on stdout */
    printf("Received a SAMPLE command: %s\n", msg.Buffer());

    /* Stores the original message */
    origMsg = msg;

    /* Send the new message */
    msgMESSAGE send;

    send.Command("TRANS");
    send.Destenv(getenv("RTAPENV"));
    send.Buffer("test buffer");
    send.Destproc("evhDummy");

    cmd.Send(send);

    /* The callback must remain installed for another command */
    return evhCB_NO_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::EndSampleCB(msgMESSAGE &msg, void *)
{
    printf("Received a reply to TRANSFER command: %s\n", msg.Buffer());
```

```
    /* Send back the reply to the command */
    if( msg.LastReply() )
        {
        origMsg.SendReply("Command completed!");
        }
    /* The callback must remain installed for another command */
    return evhCB_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::ErrSampleCB(msgMESSAGE &, void *)
{
    printf("Received an error reply to TRANSFER command\n");

    ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    if( origMsg.SendReply(ErrStack()) == FAILURE )
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    ErrStackClose();    // Error has no consequences here
                        // and can be recovered

    return evhCB_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::TimeoutSampleCB(msgMESSAGE &, void *)
{
    printf("Timeout in TRANSFER command\n");

    ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    if( origMsg.SendReply(ErrStack()) == FAILURE )
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    ErrStackClose();    // Error has no consequences here
                        // and can be recovered

    return evhCB_DELETE;
}
```

The evhDB_CMD_SEND class

The evhDB_CMD_SEND class (and subclasses) can be used to implement the same pattern.

It:

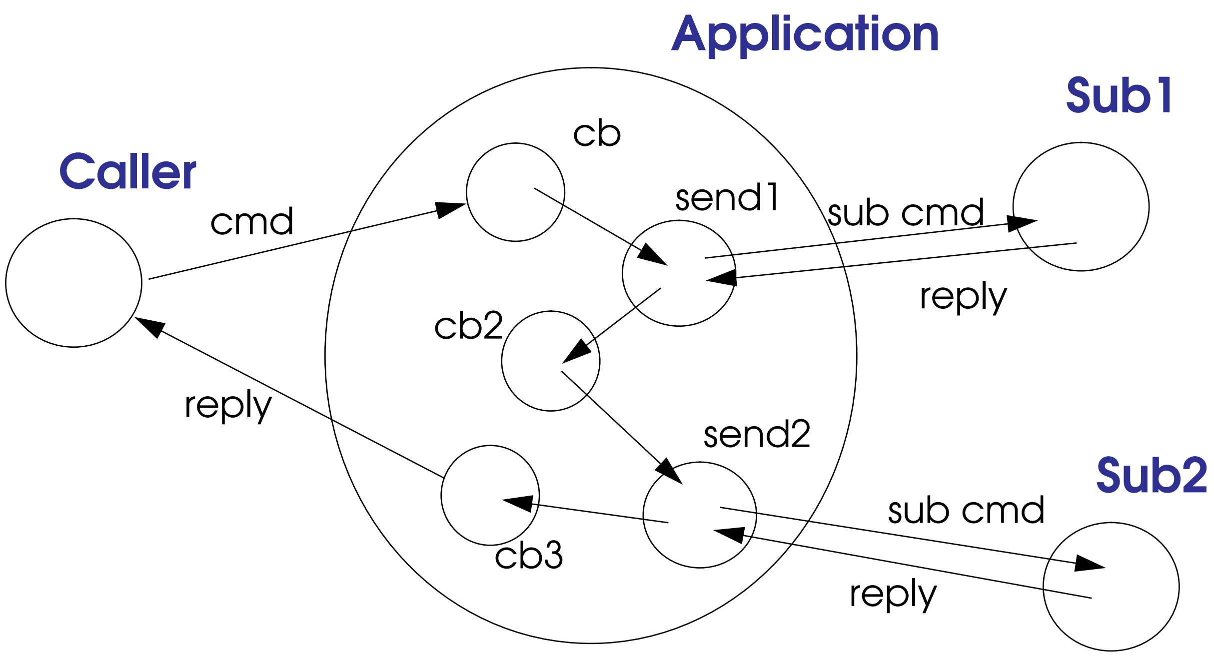
- is much more versatile
- defines a standard way of configuring applications
- allows commands' synchronization
- requires a database support point

This class should be used to implement complex patterns that involve sending sequences of commands to more than one destination process, eventually with synchronizzation among commands. Examplex are given in appendix and can be found in the design documents for TCS modules

Multi step commands

In “multi step” an application receives a command, deals with it and send one sub-command a controlled processes. When the reply comes, some new elaboration is done and a new command is sent and so on for as many steps are needed.

“Multi step” commands are implemented using the evhCOMMAND or the evhDB_CMD_SEND class.



Example 5: a multi step command

```
class APPLICATION: public evhSTD_COMMANDS
{
  public:
    APPLICATION(const dbSYMADDRESS  dbPoint);

    evhCB_COMPL_STAT  SampleCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  Step1SampleCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  EndSampleCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  ErrSampleCB(msgMESSAGE &msg, int step);
    evhCB_COMPL_STAT  TimeoutSampleCB(msgMESSAGE &msg, int step);

  private:
    evhCOMMAND cmd1;
    evhCOMMAND cmd2;
    msgMESSAGE origMsg;

}; /* end class APPLICATION */

APPLICATION::APPLICATION(const dbSYMADDRESS  dbPoint) :
    evhSTD_COMMANDS(dbPoint)
{
    evhMSG_TYPE_KEY key(msgTYPE_COMMAND);
    evhOBJ_CALLBACK cb(this,(evhCB_METHOD)SampleCB);
    ccsTIMEVAL      interval;

    evhHandler->AddCallback(key.Command("SAMPLE"),
                cb.Proc((evhCB_METHOD)SampleCB));

    interval.tv_sec  = 5;
    interval.tv_usec = 0;

    cmd1.Reply(cb.Proc((evhCB_METHOD)Step1SampleCB));
    cmd1.ErrReply(cb.Proc((evhCB_METHOD)ErrSampleCB).UserData((void*)1));
    cmd1.Timeout(cb.Proc((evhCB_METHOD)TimeoutSampleCB).UserData((void*)1),
         interval);
```

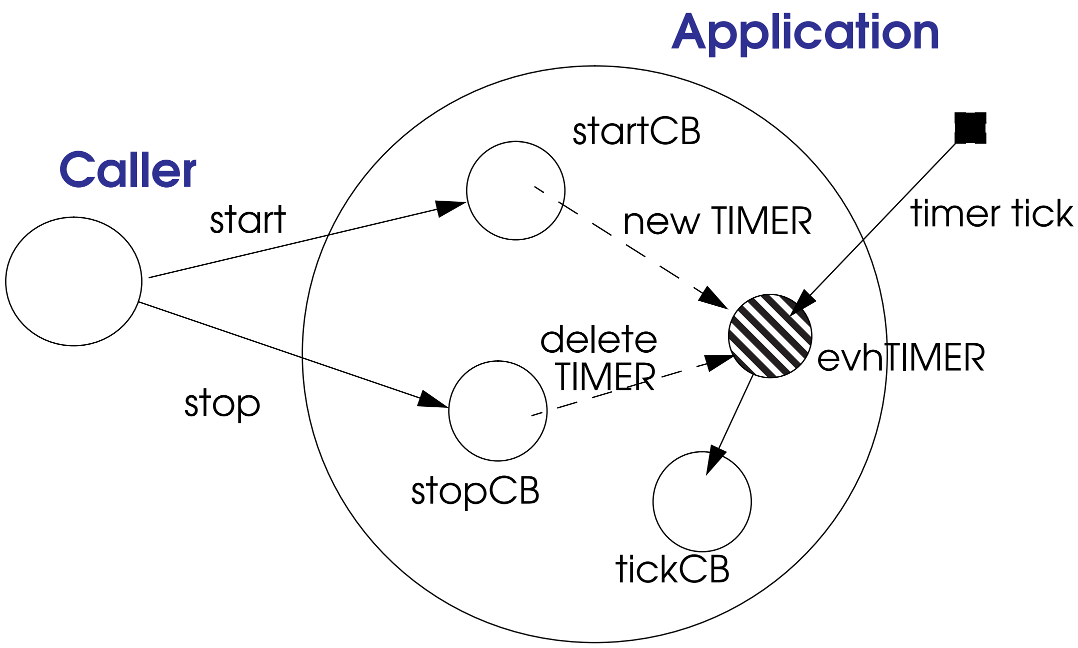
```
    cmd2.Reply(cb.Proc((evhCB_METHOD)EndSampleCB));
    cmd2.ErrReply(cb.Proc((evhCB_METHOD)ErrSampleCB).UserData((void*)2));
   cmd2.Timeout(cb.Proc((evhCB_METHOD)TimeoutSampleCB).UserData((void*)2),
            interval);
}

evhCB_COMPL_STAT  APPLICATION::SampleCB(msgMESSAGE &msg, void *)
{
    printf("Received a SAMPLE command: %s\n", msg.Buffer());

    origMsg = msg;

    msgMESSAGE send;

    send.Command("TRANS");
    send.Destenv(getenv("RTAPENV"));
    send.Buffer("test buffer");
    send.Destproc("evhDummy");

    cmd1.Send(send);

    return evhCB_NO_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::Step1SampleCB(msgMESSAGE &msg, void *)
{
    printf("Received a reply to TRANSFER command: %s\n", msg.Buffer());

    /* Send the message for the second step */
    msgMESSAGE send;

    send.Command("TRANS_2");
    send.Destenv(getenv("RTAPENV"));
    send.Buffer("test buffer");
    send.Destproc("evhDummy");

    cmd2.Send(send);
```

```
    return evhCB_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::EndSampleCB(msgMESSAGE &msg, void *)
{
    printf("Received a reply to TRANSFER_2 command: %s\n", msg.Buffer());

    /* Send back the reply to the command */
    if( msg.LastReply() )
        origMsg.SendReply("Command completed!");
    return evhCB_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::ErrSampleCB(msgMESSAGE &,  int step)
{
    printf("Received an error reply on step %d\n",step);

    ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    if( origMsg.SendReply(ErrStack()) == FAILURE )
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    ErrStackClose();
    return evhCB_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::TimeoutSampleCB(msgMESSAGE &,  int step)
{
    printf("Timeout in step %d\n",step);

    ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    if( origMsg.SendReply(ErrStack()) == FAILURE )
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    ErrStackClose();
    return evhCB_DELETE;
}
```

Cyclical timers

Another typical design pattern involves starting and stopping (for example via commands) a cyclical timer.

"Cyclical timers" are implemented using the evhTIMER class.



Example 6: Cyclical timer

```
#define _POSIX_SOURCE 1
#include "vltPort.h"
#include "ECCS.h"
#include "evhHANDLER.h"

// Include for the basic class
#include "evhCOMMAND.h"
#include "evhSTD_COMMANDS.h"
#include "evhTIMER.h"

class APPLICATION: public evhSTD_COMMANDS
{
  public:
    APPLICATION(const dbSYMADDRESS  dbPoint);

    evhCB_COMPL_STAT  TimerStartCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  TimerStopCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  TickCB(msgMESSAGE &msg, void *udata);

  private:
    evhTIMER *timer;

}; /* end class APPLICATION */

APPLICATION::APPLICATION(const dbSYMADDRESS  dbPoint) :
    evhSTD_COMMANDS(dbPoint)
{
    evhMSG_TYPE_KEY key(msgTYPE_COMMAND);
    evhOBJ_CALLBACK cb(this);

    evhHandler->AddCallback(key.Command("TSTART"),
                cb.Proc((evhCB_METHOD)TimerStartCB));
    evhHandler->AddCallback(key.Command("TSTOP"),
                cb.Proc((evhCB_METHOD)TimerStopCB));
}
```

```
evhCB_COMPL_STAT  APPLICATION::TimerStartCB(msgMESSAGE &msg, void *)
{
    printf("Received a TSTART command: %s\n", msg.Buffer());

    evhOBJ_CALLBACK cb(this,(evhCB_METHOD)TickCB);
    ccsTIMEVAL      interval;

    interval.tv_sec  = 2;

    /* If the timer already exist, return an error */
    if(timer)
    {
    ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    if( msg.SendReply(ErrStack()) == FAILURE )
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    ErrStackClose();   // Error has no consequences here
    }
    else
    {
    /* The   new evhTIMER()     call, returns a pointer to the       */
    /* newly created object. This pointer can be stored to delete    */
    /* explicitly the timeout object                                 */
    timer = new evhTIMER(cb,interval);

    /* Send back the reply to the command */
    if( msg.SendReply("Command completed!") == FAILURE )
        {
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
        ErrStackClose();   // Error has no consequences here
        }
    }
    return evhCB_NO_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::TimerStopCB(msgMESSAGE &msg, void *)
{
    printf("Received a TSTOP command: %s\n", msg.Buffer());
```

```
    /* If the timer is not running, return an error */
    if(!timer)
    {
    ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    if( msg.SendReply(ErrStack()) == FAILURE )
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
    ErrStackClose();    // Error has no consequences here
    }
    else
    {
    delete timer;
    timer = NULL;

    /* Send back the reply to the command */
    if( msg.SendReply("Command completed!") == FAILURE )
        {
        ErrAdd("eccs",eccsERR_GENERIC,__FILE_LINE__);
        ErrStackClose();    // Error has no consequences here
        }
    }
    return evhCB_NO_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::TickCB(msgMESSAGE &, void *)
{
    printf("TICK: a new tick in my timer\n");

    return evhCB_NO_DELETE;
}
```

Conclusion

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

For more details look at:

- VLT Software - CCS Event Tool Kit User Manual
  (VLT-MAN-ESO-17210-0771)

- Code examples in evh <mod>/test directory

Appendix: more examples

Example 7: evhDB_SYNC

The following example shows how to handle syncronizzation between several events (messages and changes in database value) using an instance of the evhDB_SYNC class.

```
#define _POSIX_SOURCE 1
#include "vltPort.h"

#include "ECCS.h"
#include "evhHANDLER.h"
#include "evhDB_SYNC.h"
#include "eccsTestTools.h"

#include "evhTASK.h"


class APPLICATION: public evhTASK
{
  public:
    APPLICATION();

    evhCB_COMPL_STAT  ActionCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  TimeoutCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  CommandCB(msgMESSAGE &msg, void *udata);

  private:

    // The APPLICATION class uses an instance od evhDB_SYNC to handle
    // multiple conditions
    evhDB_SYNC test;

}; /* end class APPLICATION */
```

```
APPLICATION::APPLICATION()
{
    evhMSG_TYPE_KEY key(msgTYPE_COMMAND);
    evhOBJ_CALLBACK cb(this,(evhCB_METHOD)CommandCB);

    /* Then the callback is attached to the event handler */
    evhHandler->AddCallback(key,cb);

    /*****************************************************/
    /**** Install the event handler to terminate when ****/
    /**** all the conditions are satisfied            ****/
    ccsTIMEVAL      interval;

    interval.tv_sec  = 30;
    interval.tv_usec = 0;

    // Install the callbacks for the action and the timeout
    test.Action(cb.Proc((evhCB_METHOD)ActionCB));
    test.Timeout(cb.Proc((evhCB_METHOD)TimeoutCB),interval);

    // Defines the conditions
    // Test Limit Cases:Only Commands

    test.AddCondition( key.Command("PASS1"));
    test.AddCondition( key.Command("PASS2"));
    test.AddCondition( key.Command("PASS3"));

    test.AddCondition( "Appl_data:evhTest.state", 5,TRUE);
    test.AddCondition( "Appl_data:evhTest:evhDbCmdSerial.state", 5,TRUE);

    // Activate the syncronizer
    test.Run();

    // check to see if all conditions are meet
    if ( !test.Status() )
    {
```

```
        ccsExit();
        exit(1);
    }

}

evhCB_COMPL_STAT  APPLICATION::CommandCB(msgMESSAGE &msg, void *)
{
    eccsLogComment("Received a command: %s", msg.Command());

    /* Send back the reply to the command              */
    /* (this could lead to last replies in some cases) */
    msg.LastReply(ccsTRUE);
    msg.Buffer("Command received!");
    msg.SendReply();

    /* The callback must remain installed for another command */
    return evhCB_NO_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::ActionCB(msgMESSAGE &, void *)
{
    eccsLogComment("OK, all conditions verified. I exit.");

    return evhCB_NO_DELETE | evhCB_RETURN;

}

evhCB_COMPL_STAT  APPLICATION::TimeoutCB(msgMESSAGE &, void *)
{
    /* Just print the message buffer on stdout */
    eccsLogComment("Timeout: not all the events received");

    ccsExit();
    exit(0);

    return evhCB_NO_DELETE | evhCB_RETURN;
}
```

Example 8: evhDB_CMD_SEND

The following example shows how to send a sequence of commands using the evhDB_CMD_SEND class

```
#define _POSIX_SOURCE 1
#include "vltPort.h"

#include "ECCS.h"
#include "evhHANDLER.h"
#include "eccsTestTools.h"

#include "evhDB_CMD_SEND.h"
#include "evhDB_LIST_COMMAND.h"

class SENDER: public evhDB_CMD_SEND
{
  public:

    SENDER(char *n, const dbSYMADDRESS  dbPoint,evhDB_COMMAND_TYPE type);

  protected:
    virtual evhCB_COMPL_STAT  CompletedCB(msgMESSAGE &msg, void *udata);
    virtual evhCB_COMPL_STAT  ErrorCB(msgMESSAGE &msg, void *udata);
    virtual evhCB_COMPL_STAT  TimeoutCB(msgMESSAGE &msg, void *udata);

    char *instName;

}; /* end class SENDER */

/* Body of the constructor method.              */
/* In this simple example, the constructor init */
/* everything AND send the command to the       */
/* companion SENDER                             */
SENDER::SENDER(char *name, const dbSYMADDRESS  dbPoint,
           evhDB_COMMAND_TYPE type) :
```

```
    evhDB_CMD_SEND(dbPoint,type), instName(name)
{
}

/* Body for the callbacks */
evhCB_COMPL_STAT  SENDER::CompletedCB(msgMESSAGE &msg, void *)
{
    eccsLogComment("%s - SENDER:  Received reply: %s, %s", instName,
       msg.Command(), msg.Buffer() );
    return(evhCB_DELETE);
}
evhCB_COMPL_STAT  SENDER::ErrorCB(msgMESSAGE &msg, void *)
{
    eccsLogComment("%s - SENDER:  Received error reply: %s, %s", instName,
       msg.Command(), msg.Buffer() );
    return(evhCB_DELETE);
}
evhCB_COMPL_STAT  SENDER::TimeoutCB(msgMESSAGE &, void *)
{
    eccsLogComment("%s - SENDER:  Timeout expired", instName);
    return(evhCB_DELETE);
}

/****************************************/
/* Definition for the APPLICATION class   */
/****************************************/

class APPLICATION: public evhDB_TASK
{
  public:
    APPLICATION(const dbSYMADDRESS  dbPoint);
    ~APPLICATION();

    evhCB_COMPL_STAT  ExecuteCB(msgMESSAGE &msg, void *udata);
    evhCB_COMPL_STAT  ResetCB(msgMESSAGE &msg, void *udata);

  private:
    SENDER *send1;
```

```
    SENDER *send2;
    SENDER *send3;

}; /* end class APPLICATION */

APPLICATION::APPLICATION(const dbSYMADDRESS  dbPoint) : evhDB_TASK(dbPoint)
{
    evhMSG_TYPE_KEY key(msgTYPE_COMMAND);
    evhOBJ_CALLBACK cb(this);

    evhHandler->AddCallback(key.Command("EXE"),
                cb.Proc((evhCB_METHOD)ExecuteCB));
    evhHandler->AddCallback(key.Command("RESET"),
                cb.Proc((evhCB_METHOD)ResetCB));

    /* Allocates the test command instances */
    dbSYMADDRESS  dbSubPoint;
    sprintf(dbSubPoint,"%s:%s", dbPoint,"evhDbCmdSend_1");
    send1 = new SENDER("evhDbCmdSend_1",dbSubPoint,evhDB_COMMAND_CLASS);
    sprintf(dbSubPoint,"%s:%s", dbPoint,"evhDbCmdSend_2");
    send2 = new SENDER("evhDbCmdSend_2",dbSubPoint,evhDB_COMMAND_CLASS);
    sprintf(dbSubPoint,"%s:%s", dbPoint,"evhDbCmdSend_3");
    send3 = new SENDER("evhDbCmdSend_3",dbSubPoint,evhDB_COMMAND_CLASS);
}

APPLICATION::~APPLICATION()
{
    if(send1) delete send1;
    if(send2) delete send2;
    if(send3) delete send3;
}

evhCB_COMPL_STAT  APPLICATION::ExecuteCB(msgMESSAGE &msg, void *)
{
    ccsCOMPL_STAT stat = SUCCESS;

    eccsLogComment("Received an EXE command: %s", msg.Buffer());
```

```
    send1->Timeout(1);
    send2->Timeout(1);
    send3->Timeout(1);

    /* Execute the sequence                         */
    /* The 3 commands must be executed in a sequence */
    evhDB_SEND_ID id;
    stat = send1->Send("SEND1", msg.Buffer(),msg.Buflen(),&id);
    stat = send2->Send("SEND2", msg.Buffer(),msg.Buflen(),&id,id);
    stat = send3->Send("SEND3", msg.Buffer(),msg.Buflen(),&id,id);


    /* Send back the reply to the command */
    msg.LastReply(ccsTRUE);
    msg.Buffer("Command received!");
    msg.SendReply();

    /* The callback must remain installed for another command */
    return evhCB_NO_DELETE;
}

evhCB_COMPL_STAT  APPLICATION::ResetCB(msgMESSAGE &msg, void *)
{
    eccsLogComment("Received an RESET command: %s", msg.Buffer());

    send1->Reset();
    send2->Reset();
    send3->Reset();

    /* Send back the reply to the command */
    msg.LastReply(ccsTRUE);
    msg.Buffer("Command received!");
    msg.SendReply();

    return evhCB_NO_DELETE;
}
```