Contents

Chapter 1

PMCommand.h

```
#ifndef _PMcommand_H_
#define _PMcommand_H_


#include "testFMCMDevice.h"

namespace testFMCM {

class PMcommand{
public:
static void  GetPMData(testFMCMDevice * pDev);
private:
static char  PMCommandData[21];
static void  SetPMCommandData(testFMCMDevice * pDev);
static void  SendPMcommand(testFMCMDevice * pDev);
static int  ReceivePM(testFMCMDevice * pDev);
static int   ReceiveTrailerCheck(int fd, testFMCMDevice * pDev);
static int    ReceiveData(int fd, testFMCMDevice * pDev);
static int  getbit(int p, char x);
static bool  ResponseChksum(char * responseHeader,
   testFMCMDevice * pDev,int buffersize);
static int    EMCtestPM(testFMCMDevice * pDev);
};
};
#endif
%
```

Chapter 2

PMCommand.cpp

```
#include "PMcommand.h"
#include "testFMCMRealtime.h"
#include "testFMCMGlobalStore.h"

using namespace testFMCM;
char PMcommand:: PMCommandData[21];



/*PMcommand::GetPMData
 *
 *Function which instantiates the sending and receiving of PM data
 * ******************/

void PMcommand::GetPMData(testFMCMDevice * pDev){

int PMcounter= 0;
SendPMcommand(pDev);
testFMCMRT::Sleep(2,0);
while (ReceivePM(pDev) == 1 && PMcounter<5){
SendPMcommand(pDev);
testFMCMRT::Sleep(2,0);
PMcounter++;
}
pDev->NumberOfRetransmissions.set(pDev->NumberOfRetransmissions.get() + PMcounter);
pDev->PmRetransmitted.set(pDev->PmRetransmitted.get() + PMcounter);
if(PMcounter>=5){
pDev->FailedTransmissions.set(pDev->FailedTransmissions.get() + 1);
pDev->PmFailed.set(pDev->PmFailed.get() + 1);
}
/*Remove for EMC test with FMCM FPGA 7*/
// else{
// EMCtestPM(pDev);
// }
/************************************/
```

```
}



/*PMcommand::SendPMcommand
 *
 *Function which writes the PM command to the RS-422 interface and the FMCM
 * ***********************/

void PMcommand::SendPMcommand(testFMCMDevice * pDev){
int fd = -1;
int sentcnt = 0;
int SentDataSize= 0;

fd = pDev->fd.get();
SetPMCommandData(pDev);
SentDataSize= strlen(PMCommandData);
testFMCMRT::Flush(fd);
sentcnt= testFMCMRT::Write(fd,PMCommandData);
if(sentcnt!= SentDataSize ){
if(sentcnt==-1){
}
}
else{
pDev->PmSent.set(pDev->PmSent.get() + 1);
}
/* setting data in the Fesa server*/

/* For EMC test!*/
for(unsigned int i=0;i<10;i++){
pDev->SynchronizationCommand.setCell(i,(short)PMCommandData[i]);
}
pDev->StartOfCommand.set(PMCommandData[10]);
pDev->CommandCode.set((short)PMCommandData[11]);
for(unsigned int i=0; i< 6; i++ ){
pDev->CommandArg.setCell(i, (short)PMCommandData[i+12]);
}
for(unsigned int i=0;i<2;i++){
pDev->CommandChecksum.setCell(i, (short)PMCommandData[i+18]);
}
/*********************************************************/
}
```

```
/*PMcommand::SetPMCommandData
 *
 *Function which builds the PM command in a vector called PMCommandData
 * ************************/

void PMcommand::SetPMCommandData(testFMCMDevice * pDev){
short lsb, msb, checksum= 0;

for (unsigned int i=0; i<10; i++){
PMCommandData[i]= 0x0D ; // Synchronization
}
PMCommandData[10]= 0x2A; // Start of command
PMCommandData[11]= 0x70; // PM command p
PMCommandData[12]= pDev->PMtype.get(); // Command arguments 6 bytes :
PMCommandData[13]= 0x30;
PMCommandData[14]= 0x30;
PMCommandData[15]= 0x30;
PMCommandData[16]= 0x30;
PMCommandData[17]= 0x30;
for(unsigned int i=11;i<18;i++){
checksum = checksum + (int)PMCommandData[i];
}
checksum = checksum + 0x55aa;
lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF ;
PMCommandData[18]= msb;
PMCommandData[19]= lsb;
PMCommandData[20]= '\0';
}



/*PMcommand::ReceivePM
 *
 *Function which reads the PM data set from the
 *RS-422 interface and the FMCM.
 *It reads first the Response Header,
 *then the PM data, then the checksum and the trailer, and
 *in the end returns 0 if checksum was correct.
 *subfunctions:  ReceiveData(), SimpleReceive(), ResponseChksum()
 *
 *Return 0 = success, Return 1 = error
 * ******************/

int PMcommand::ReceivePM(testFMCMDevice * pDev){

unsigned int buffersize = 28; // Expected size of Receive Header is 28 bytes
char rcvbuf[buffersize];
int fd = -1;
```

```
unsigned int rcvcnt= 0;

/* Read the Response Header:*/
fd= pDev->fd.get();
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
if(rcvcnt< buffersize){
pDev->ComErrorByteCount.set(pDev->ComErrorByteCount.get() + 4032 -14);
pDev->MissingByteCountHeader.set(pDev->MissingByteCountHeader.get()
+ (buffersize - rcvcnt));
if(rcvcnt==0){
}
testFMCMRT::Flush(fd);
return 1;
}
if(rcvcnt==buffersize){
/*Sets the different fields in the Receive Header if read was sucessful:*/
pDev->SynchronizationResponse.set(rcvbuf[0]);
pDev->StartOfCommandReturned.set(rcvbuf[1]);
pDev->CommandCodeReturned.set(rcvbuf[2]);
for(unsigned int i=0; i< 6; i++ ){
pDev->CommandArgReturned.setCell(i, rcvbuf[i+3]);
}
for(unsigned int i=0;i<2;i++){
pDev->CommandChecksumReturned.setCell(i, rcvbuf[i+9]);
}
for(unsigned int i=0;i<8;i++){
pDev->Errorbits.setCell(i, getbit(7-i,rcvbuf[11]));
}
for(unsigned int i=0;i<7;i++){
pDev->CurrentTimestamp.setCell(i, rcvbuf[i+12]);
}
for(unsigned int i=0;i<8;i++){
pDev->InfoBits.setCell(i, getbit(7-i,rcvbuf[19]));
}
/*ENDURANCE RUN */
for(unsigned int i=0;i<7;i++){
pDev->TimestampLastPM.setCell(i, rcvbuf[i+12]);
}
/*********************************************/
// for(unsigned int i=0;i<7;i++){
// pDev->TimestampLastPM.setCell(i, rcvbuf[i+20]);
// }
pDev->Spare.set(rcvbuf[27]);
/*Check the type of command being sent, read the rest of the message and
* set the fields:*/
if (rcvbuf[2] == 'p'){
if(ReceiveData(fd,pDev) == 0){
if(ReceiveTrailerCheck(fd,pDev)== 0){
/*EMC test with FMCM FPGA 7*/
```

```
if (EMCtestPM(pDev)==1){
pDev->NumberOfRetransmissions.set(pDev->NumberOfRetransmissions.get() + 1);
testFMCMRT::Flush(fd);
return 1;
}
// if (ResponseChksum(rcvbuf,pDev,buffersize)== true){
// }
// else{
// return 1;
// }
}
else{
testFMCMRT::Flush(fd);
return 1;
}
}
else{
testFMCMRT::Flush(fd);
return 1;
}
}
else{
pDev->ComErrorByteCount.set(pDev->ComErrorByteCount.get() + 4032 -14);
testFMCMRT::Flush(fd);
return 1;
}

// //EMC test with 512 byte:
//
// if (ReceiveData(fd, pDev) == 1){
// return 1;
// }
}
pDev->PmReceived.set(pDev->PmReceived.get() + 1);
testFMCMRT::Flush(fd);
return 0;
}



/*Function getbit
 *
 * Get bit number p from a char(8 bits), counting from the right.
 * 0=<p>8
 * ************************************************************************/

int PMcommand::getbit(int p, char x){
return((x >> (p)) & ~(~0 << 1));
}
```

```
/*PMcommand::ReceiveData
 *
 *Function which reads the PM data. 4000 bytes.
 *Return 0 = sucessfull, Return 1= Error
 * ********************/

int   PMcommand::ReceiveData(int fd, testFMCMDevice * pDev){
unsigned int buffersize = 4000;
char rcvbuf[buffersize];
unsigned int rcvcnt=0;

rcvcnt=testFMCMRT::Read(fd, rcvbuf, buffersize);
if(rcvcnt< buffersize){
pDev->ComErrorByteCount.set(pDev->ComErrorByteCount.get() + 4032 -14);
pDev->MissingByteCountData.set(pDev->MissingByteCountData.get()+ (buffersize - rcvcnt));
if(rcvcnt==0){
}
return 1;
}
if(rcvcnt==buffersize){
for(unsigned int i = 0; i<buffersize; i++){   // 2000 samples
pDev->PM.setCell(i, rcvbuf[i]);
}
/*Receiving PM data from FMCM FPGA 7*/
int counter=0;
for(unsigned int i = 0; i<buffersize; i=i+2){   // 2000 samples
long longPM1= (long)pDev->PM.getCell(i+1);
  long longPM2 = ((long)pDev->PM.getCell(i))<<8;
  long longPM = longPM1+ longPM2;
  pDev->PmData2000.setCell(counter, longPM);
  counter++;
}
/*****************************************/
}
return 0;
}



/*PMcommand::ReceiveTrailerCheck
 *
 *Function to read the checksum and the trailer of the PM message: 4 bytes
 *Return 0 = success , Return 1 = error
 * ***************************/

int PMcommand::ReceiveTrailerCheck(int fd, testFMCMDevice* pDev){
```

```
int buffersize = 4; // Expected size is 4 bytes
char rcvbuf[buffersize];
int rcvcnt=0;
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
if(rcvcnt< buffersize){
pDev->ComErrorByteCount.set(pDev->ComErrorByteCount.get() + 4032 -14);
pDev->MissingByteCountTrailerCheck.set(pDev->MissingByteCountTrailerCheck.get()
+ (buffersize - rcvcnt));
if(rcvcnt==0){
}
return 1;
}
if(rcvcnt==buffersize){
pDev->AccumBytecount.set(pDev->AccumBytecount.get() + 4032 - 14);
for(unsigned int i=0;i<2;i++){
pDev->ResponseChecksum.setCell(i, rcvbuf[i+0]);
}
for(unsigned int i=0;i<2;i++){
pDev->ResponseTrailer.setCell(i, rcvbuf[i+2]);
}
}
return 0;
}



/*PMcommand::ResponseChksum
 *
 *Function which calculates the Checksum of the entire PM command.
 *Returns true if calculated checksum is identical with received checksum.
 * *************************/

bool PMcommand::ResponseChksum(char * responseHeader,
testFMCMDevice * pDev,int buffersize){

unsigned int size = buffersize;
short checksum =0;
short lsb, msb = 0;

for (unsigned int i= 1; i<size;i++){
checksum = checksum + responseHeader[i];
}
checksum = checksum + 0x55aa;
lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF; //most significant bits..

if ((msb == pDev->ResponseChecksum.getCell(0))
&& ( lsb== pDev->ResponseChecksum.getCell(1))){
```

```
  return true;
}
else {
return false;
}
}



/*PMcommand:: EMCtestPM
 *
 *Function which counts all erronous bytes that h
 *ave been succesfully received
 * Returns 1 : byte error, Returns 0: No byte errors
 * ******************/

int PMcommand:: EMCtestPM(testFMCMDevice * pDev){

long errorcount= 0;
//CHECKING RESPONSE HEADER:
if(pDev->Spare.get() != 0 ){
errorcount++;
}

//As Infobits are not implemented yet they are set to zero:
if( pDev->InfoBits.getCell(0) != 0 || pDev->InfoBits.getCell(1) != 0 ||
 pDev->InfoBits.getCell(2) != 0 || pDev->InfoBits.getCell(3) != 0 ||
 pDev->InfoBits.getCell(4) != 0 || pDev->InfoBits.getCell(5) != 0 ||
  pDev->InfoBits.getCell(6) != 0 || pDev->InfoBits.getCell(7) != 0){
errorcount++;
}

//As Errorbits are not implemented yet they are set to zero:
if( pDev->InfoBits.getCell(0) != 0 || pDev->InfoBits.getCell(1) != 0 ||
 pDev->InfoBits.getCell(2) != 0 || pDev->InfoBits.getCell(3) != 0 ||
 pDev->InfoBits.getCell(4) != 0 || pDev->InfoBits.getCell(5) != 0 ||
  pDev->InfoBits.getCell(6) != 0 || pDev->InfoBits.getCell(7) != 0){
errorcount++;
}

for(unsigned int i = 0; i<2; i++){
if(pDev->CommandChecksumReturned.getCell(i)!= pDev->CommandChecksum.getCell(i)){
errorcount++;
}
}

for(unsigned int i = 0; i<6; i++){
if (pDev->CommandArgReturned.getCell(i)!= pDev->CommandArg.getCell(i)){
errorcount++;
```

```
}
}

if(pDev->CommandCodeReturned.get()!= pDev->CommandCode.get()){
errorcount++;
}

if(pDev->StartOfCommandReturned.get()!= pDev->StartOfCommand.get()){
errorcount++;
}

if(pDev->SynchronizationResponse.get()!= 0x0d){
errorcount++;
}

//CHECKING PM DATA

/*VERSION 1: FMCM simulator(counts from 1 to 256, 15 times and from 1 to 160 1 time.
//for(unsigned int i= 0; i<15; i++){
// for(unsigned int j=0; j<256; j++){
// if (pDev->PM.getCell(j) != j+1){
// errorcount++;
// }
// }
//}
//for(unsigned int i= 0; i<160; i++){
// if (pDev->PM.getCell(i) != i+1){
// errorcount++;
// }
//}

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

/*VERSION 2: FMCM FPGA 7(counts from 0 to 1999,*/
int i = 0;
for(unsigned int j=0; j<2000; j=j+1){

if (pDev->PmData2000.getCell(j)!= i){
errorcount++;
}
i++;
}

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

/*VERSION 3: BUFFER 512 bytes*/
//
//int i = 0;
//for(unsigned int j=1; j<484; j=j+2){
```

```
// if (pDev->PM.getCell(j) != i){
// errorcount++;
// }
// if(pDev->PM.getCell(j-1)!= 0){
// errorcount++;
// }
// i++;
//}
/*************************************/
//CHECKING CHECKSUM AND TRAILER:


/*FMCM FPGA 7: checksum is not implemented yet and set to null*/
for(unsigned int i = 0; i<2; i++){
if (pDev->ResponseChecksum.getCell(i)!= 0){
errorcount++;
}
}
if (pDev->ResponseTrailer.getCell(0)!= 0x3c){ errorcount++; }
if (pDev->ResponseTrailer.getCell(1)!= 0x3e){ errorcount++; }
pDev->GeneralByteErrorCount.set(pDev->AccumErrorcount.get()+
pDev->MissingByteCountTrailerCheck.get()+ pDev->MissingByteCountData.get() +
pDev->MissingByteCountHeader.get() );

//UPDATING ACCUMULATED ERROR COUNT:
pDev->AccumErrorcount.set(pDev->AccumErrorcount.get() + errorcount);

/*FMCM FPGA 7*/
if(errorcount> 0){
return 1;
}
/**************************/

return 0;
}
```

Chapter 3

StatusCommand.h

```
//
//  FESA framework June 2004.
//
// Use this code as a starting-point to develop your own equipment class

#ifndef _testFMCM_StatusCommand_H_
#define _testFMCM_StatusCommand_H_

#include <fesa/Fesa.h>
#include "testFMCMDevice.h"
#include "testFMCMGlobalStore.h"

namespace testFMCM {

class StatusCommand: public RTAction <RTEvent, testFMCMGlobalStore, testFMCMDevice >
public:
void execute(RTEvent *);
StatusCommand(const string& name, AbstractRTAction::RTActionConfig& rtActCfg) ;
private:
void  SetStatusCommandData();
char  StatusCommandData[21];
void  SendStatusCommand();
int  ReceiveStatus(MultiplexingContext* ctx );
int  ReceiveTrailerCheck(int fd, testFMCMDevice* pDev);
int  ReceiveData(int fd, testFMCMDevice * pDev);
int  getbit(int p, char x);
bool  ResponseChksum(char * responseHeader,
testFMCMDevice * pDev,int buffersize);
void  NotifyPropertyPM(testFMCMDevice * pDev,MultiplexingContext* ctx);
void  PMtrigger(testFMCMDevice * pDev,MultiplexingContext* ctx);
};
}
#endif
```

Chapter 4

StatusCommand.cpp

```
//
//  FESA framework June 2004.
//
// Use this code as a starting-point to develop your own equipment class

#include <StatusCommand.h>
#include "testFMCMRealtime.h"
#include "PMcommand.h"


// INPUT fields:
// OUTPUT fields:


using namespace testFMCM;

StatusCommand::StatusCommand(const string& name,
 AbstractRTAction::RTActionConfig& rtActCfg) :
RTAction<RTEvent, testFMCMGlobalStore, testFMCMDevice>(name, rtActCfg){}

/*StatusCommand::execute
 *
 *Function which instaniates the sending and receiving of Status data
 * */
void StatusCommand::execute(RTEvent * pEv){

MultiplexingContext* ctx = pEv->getMultiplexingContext();
log << "executing RT action: testFMCMStatusCommand"<<endInfo;
testFMCMDevice * pDev = deviceCollection[0];
int Statuscounter = 0;
SendStatusCommand();
testFMCMRT::Sleep(0,120);
while (ReceiveStatus(ctx ) == 1 && Statuscounter<3){
SendStatusCommand();
```

```
testFMCMRT::Sleep(0,120);
Statuscounter++;
}
pDev->StatusRetransmitted.set(pDev->StatusRetransmitted.get() + Statuscounter);
if(Statuscounter>=3){
pDev->StatusFailed.set(pDev->StatusFailed.get() + Statuscounter);
}

}


/*StatusCommand::SendStatusCommand
 *
 *Function which writes the Status command
 *to the RS-422 interface and the FMCM
 * */
void StatusCommand::SendStatusCommand(){

int fd = -1;
int sentcnt = 0;
for (unsigned int i=0; i < deviceCollection.size(); i++){
testFMCMDevice * pDev = deviceCollection[i];
fd = pDev->fd.get();
        SetStatusCommandData();
        int SentDataSize= strlen(StatusCommandData);
        testFMCMRT::Flush(fd);
sentcnt= testFMCMRT::Write(fd,StatusCommandData);
if(sentcnt!= SentDataSize ){
   if(sentcnt==-1){
  cout<<pDev->name.get()<<"(fd"<<fd<<"):
  writing Status Command request failed with:" ; cout.flush();
   }
}
else{
cout<< "Sending Status Command to "<<pDev->name.get()
<<" , fd = "<<fd<<"    Status Command: "<<endl;
pDev->StatusSent.set(pDev->StatusSent.get() + 1);
}
// /* setting data in the Fesa server*/
// for(unsigned int i=0;i<10;i++){
// pDev->SynchronizationCommand.setCell(i,(short)StatusCommandData[i]);
// }
// pDev->StartOfCommand.set(StatusCommandData[10]);
// pDev->CommandCode.set((short)StatusCommandData[11]);
// for(unsigned int i=0; i< 6; i++ ){
// pDev->CommandArg.setCell(i, (short)StatusCommandData[i+12]);
// }
//
// for(unsigned int i=0;i<2;i++){
```

```
// pDev->CommandChecksum.setCell(i, (short)StatusCommandData[i+18]);
// }
}
}


/*StatusCommand::SetStatusCommandData
 *
 *Function which builds the Status command in
 *a vector called StatusCommandData
 * */
void StatusCommand::SetStatusCommandData() {

short lsb, msb, checksum= 0;
for (unsigned int i=0; i<10; i++){
StatusCommandData[i]= 0x0D ; // Synchronization
}
StatusCommandData[10]= 0x2A; // Start of command
StatusCommandData[11]= 0x73; // Status command s
StatusCommandData[12]= 0x30; // Command arguments 6 bytes :
StatusCommandData[13]= 0x30;
StatusCommandData[14]= 0x30;
StatusCommandData[15]= 0x30;
StatusCommandData[16]= 0x30;
StatusCommandData[17]= 0x30;
for(unsigned int i=11;i<18;i++){
checksum = checksum + (int)StatusCommandData[i];
}
checksum = checksum + 0x55aa;
    lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF ;
StatusCommandData[18]= msb;
StatusCommandData[19]= lsb;
StatusCommandData[20]= '\0';


}

/*StatusCommand::ReceiveStatus
 *
 *Function which reads the Status data set from
 * the RS-422 interface and the FMCM.
 *It reads first the Response Header,
 *then the Status data,
 * then the checksum and the trailer, and
 *in the end returns 0 if checksum was correct.
 *subfunctions:  ReceiveData(), SimpleReceive(), ResponseChksum()
 *
 *Return 0 = success, Return 1 = error
```

```
 * */
int StatusCommand::ReceiveStatus(MultiplexingContext* ctx ){

unsigned int buffersize = 28; // Expected size of Receive Header is 28 bytes
char rcvbuf[buffersize];
int fd = -1;
unsigned int rcvcnt= 0;
for (unsigned int i=0; i < deviceCollection.size(); i++){
testFMCMDevice * pDev = deviceCollection[i];
fd= pDev->fd.get();
    /* Read the Response Header:*/
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
// for(unsigned int i=0;i<rcvcnt;i++){
// cout<<i<<": "<<rcvbuf[i]<<endl;
// }
if(rcvcnt< buffersize){
cout<<"Error reading Response Header: rcvcnt<buffersize"<<rcvcnt<<endl;
if(rcvcnt==0){
cout<<"Error reading Response Header: End of line"<<endl;
}
testFMCMRT::Flush(fd);
return 1;
}
if(rcvcnt==buffersize){
/*Sets the different fields in the Receive Header if read was sucessful:*/
//cout<<"Reading Response Header"<<endl;
pDev->SynchronizationResponse.set(rcvbuf[0]);
pDev->StartOfCommandReturned.set(rcvbuf[1]);
pDev->CommandCodeReturned.set(rcvbuf[2]);
for(unsigned int i=0; i< 6; i++ ){
pDev->CommandArgReturned.setCell(i, rcvbuf[i+3]);
}
for(unsigned int i=0;i<2;i++){
pDev->CommandChecksumReturned.setCell(i, rcvbuf[i+9]);
}
  for(unsigned int i=0;i<8;i++){
  short temp = rcvbuf[11];
pDev->Errorbits.setCell(i, getbit(7-i,temp));
}
for(unsigned int i=0;i<7;i++){
pDev->CurrentTimestamp.setCell(i, rcvbuf[i+12]);
}
for(unsigned int i=0;i<8;i++){
short temp = rcvbuf[11];
pDev->InfoBits.setCell(i, getbit(7-i,temp));
}
    /****ENDURANCE RUN ****************************/
for(unsigned int i=0;i<7;i++){
pDev->TimestampLastPM.setCell(i, rcvbuf[i+12]);
```

```
}
/*********************************************/
// for(unsigned int i=0;i<7;i++){
// pDev->TimestampLastPM.setCell(i, rcvbuf[i+20]);
// }
if(rcvbuf[27]==0x00){
pDev->Spare.set(0x30);
}
else{
pDev->Spare.set(rcvbuf[27]);
}
/*Check the type of command being sent,
read the rest, set the fields and update the property: */
if (rcvbuf[2] == 's'){
if(ReceiveData(fd,pDev) == 0){
if(ReceiveTrailerCheck(fd,pDev)== 0){
// if (ResponseChksum(rcvbuf,pDev,buffersize)== true){
// cout<<rcvbuf[2]<<" Response received"<<endl;
// }
// else{
// return 1;
// }
}
else{
return 1;
}
}
else{
return 1;
}
}
else{
return 1;
}
testFMCMRT::Flush(fd);
}
cout<<"status message received"<<endl;
pDev->StatusReceived.set(pDev->StatusReceived.get() + 1);
PMtrigger(pDev,ctx);
log << "StatusCommand operates on device "
<< pDev->name.get()
<< endInfo;
}
return 0;
}

/*Function getbit
 *
 * Get bit number p from a char(8 bits), counting from the right.
```

```
 * 0=<p>8
 * ************************************************************************/
int StatusCommand::getbit(int p, char x){

return((x >> (p)) & ~(~0 << 1));
}

/*StatusCommand::ReceiveTrailerCheck
 *
 * Function to read the remaining parts of the status Message from the FMCM
 * Assumes Receive Header has already been read
 * Data:0 bytes, Response Checksum: 2 bytes, Trailer: 2 bytes
 *
 * Return 0 = success, Return 1 = error
 * */
int StatusCommand::ReceiveTrailerCheck(int fd, testFMCMDevice* pDev){

int buffersize = 4; // Expected size is 4 bytes
char rcvbuf[buffersize];
int rcvcnt=0;
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
    if(rcvcnt< buffersize){
cout<<"Error reading Trailer and Checksum: rcvcnt<buffersize"<<rcvcnt<<endl;
if(rcvcnt==0){
cout<<"Error reading Trailer and Checksum: End of line"<<endl;
}
return 1;
}
if(rcvcnt==buffersize){
// for (unsigned i=0; i<4; i++){
// cout<< "trailcheck"<<(i) << ": "<< rcvbuf[i]<<endl;
// }
for(unsigned int i=0;i<2;i++){
pDev->ResponseChecksum.setCell(i, rcvbuf[i+0]);
}
for(unsigned int i=0;i<2;i++){
pDev->ResponseTrailer.setCell(i, rcvbuf[i+2]);
}
}
return 0;
}

/*StatusCommand::ResponseChksum
 *
 *Function which calculates the Checksum of the entire PM command.
 *Returns true if calculated checksum is identical with received checksum.
 * */
bool StatusCommand::ResponseChksum(char * responseHeader,
testFMCMDevice * pDev,int buffersize){
```

```
unsigned int size = buffersize;
short checksum =0;
short lsb, msb = 0;
for (unsigned int i= 1; i<size;i++){
checksum = checksum + responseHeader[i];
}
checksum = checksum + 0x55aa;
lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF;
if ((msb == pDev->ResponseChecksum.getCell(0)) &&
( lsb== pDev->ResponseChecksum.getCell(1))){
  return true;
}
else {
return false;
}
}

/*StatusCommand::ReceiveData
 *
 *Function which reads the Status data. 32 bytes.
 *Return 0 = sucessfull, Return 1= Error
 * */
int StatusCommand::ReceiveData(int fd, testFMCMDevice * pDev){
int buffersize = 32; // Expected size is 32 bytes
char rcvbuf[buffersize];
int rcvcnt=0;

rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
if(rcvcnt< buffersize){
cout<<"Error reading Status Data: rcvcnt<buffersize"<<rcvcnt<<endl;
if(rcvcnt==0){
cout<<"Error reading Status Data: End of line"<<endl;
}
return 1;
}
if(rcvcnt==buffersize){
pDev->FPGAconfig.set(rcvbuf[0]);
for(unsigned int i=0;i<3;i++){
pDev->UpTimeMinutes.setCell(i, rcvbuf[1+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->PreAlarmTreshPotentiometer.setCell(i, rcvbuf[4+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->AlarmTreshPotentiometer.setCell(i, rcvbuf[6+i]);
}
for(unsigned int i=0;i<2;i++){
```

```
pDev->AlarmCounter.setCell(i, rcvbuf[8+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->PreAlarmCounter.setCell(i, rcvbuf[10+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->ActualMagnetVoltage.setCell(i, rcvbuf[12+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->Uext.setCell(i, rcvbuf[14+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->IDiffSim.setCell(i, rcvbuf[16+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->IDiffDCCT.setCell(i, rcvbuf[18+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->MinFieldDev.setCell(i, rcvbuf[20+i]);
}
for(unsigned int i=0;i<2;i++){
pDev->MaxFieldDev.setCell(i, rcvbuf[22+i]);
}
for(unsigned int i=0;i<4;i++){
pDev->DiffTimestamp.setCell(i, rcvbuf[24+i]);
}
for(unsigned int i=0;i<7;i++){
pDev->IDandConfigStatus.setCell(i, getbit(7-i, rcvbuf[28]));
}
for(unsigned int i=0;i<7;i++){
pDev->DeviceStatus.setCell(i, getbit(7-i, rcvbuf[29]));
}
for(unsigned int i=0;i<2;i++){
pDev->DobbelSpare.setCell(i, rcvbuf[30+i]);
}
}
return 0;
}

/*StatusCommand::NotifyPropertyPM
 *
 * */
void StatusCommand::NotifyPropertyPM(testFMCMDevice * pDev,MultiplexingContext* ctx){

std:: string className = pGlobalStore->name.get();
std:: string notificationstr;
notificationstr = "PmData:";
notificationstr += pDev->name.get();
AbstractEquipmentRT::notify(ctx,className, notificationstr);
```

```
}

/*StatusCommand::PMtrigger(
 *
 *If timestamp of last PM aquisition has canged, g
 *et PM data, update the PMTimestamp field
 *and notify the property PMdata
 * */
 void  StatusCommand::PMtrigger(testFMCMDevice * pDev,
 MultiplexingContext* ctx){

    /*ENDURANCE RUN*/
  if (pDev->TimestampLastPM.getCell(2)!=pDev->PMTimestamp.getCell(2)){
PMcommand::GetPMData(pDev);
for(unsigned int i = 0; i<7; i++){
pDev->PMTimestamp.setCell(i,pDev->TimestampLastPM.getCell(i));
}
NotifyPropertyPM(pDev, ctx);
}

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

/* if (pDev->TimestampLastPM.getCell(0)!=pDev->PMTimestamp.getCell(0) ||
pDev->TimestampLastPM.getCell(1)!=pDev->PMTimestamp.getCell(1) ||
pDev->TimestampLastPM.getCell(2)!=pDev->PMTimestamp.getCell(2) ||
pDev->TimestampLastPM.getCell(3)!=pDev->PMTimestamp.getCell(3) ||
pDev->TimestampLastPM.getCell(4)!=pDev->PMTimestamp.getCell(4) ||
pDev->TimestampLastPM.getCell(5)!=pDev->PMTimestamp.getCell(5) ||
pDev->TimestampLastPM.getCell(6)!=pDev->PMTimestamp.getCell(6)){
PMcommand::GetPMData(pDev);
for(unsigned int i = 0; i<7; i++){
pDev->PMTimestamp.setCell(i,pDev->TimestampLastPM.getCell(i));
}
NotifyPropertyPM(pDev, ctx);
}*/

 }
```

Chapter 5

UTCCommand.h

```
//
//  FESA framework June 2004.
//
//  Use this code as a starting-point to develop your own equipment class

#ifndef _testFMCM_UTCcommand_H_
#define _testFMCM_UTCcommand_H_

#include <fesa/Fesa.h>
#include "testFMCMDevice.h"
#include "testFMCMGlobalStore.h"


namespace testFMCM {

class UTCcommand: public RTAction <RTEvent, testFMCMGlobalStore, testFMCMDevice > {
public:
void  execute(RTEvent *);
UTCcommand(const string& name, AbstractRTAction::RTActionConfig& rtActCfg) ;
private:
void  SetUTCCommandData();
char  UTCCommandData[21];
void  SendUTC();
int  ReceiveUTC(MultiplexingContext* ctx);
int  ReceiveTrailerCheck(int fd, testFMCMDevice * pDev);
int  getbit(int p, char x);
bool  ResponseChksum(char * responseHeader,
testFMCMDevice * pDev,int buffersize);
void  NotifyPropertyPM(testFMCMDevice * pDev,
MultiplexingContext* ctx);
void  PMtrigger(testFMCMDevice * pDev,MultiplexingContext* ctx);
void  CheckTime(testFMCMDevice * pDev);
};
}
#endif
```

Chapter 6

UTCCommand.cpp

```
//
//  FESA framework June 2004.
//
// Use this code as a starting-point to develop your own equipment class

#include <UTCcommand.h>
#include "testFMCMRealtime.h"
#include "PMcommand.h"
#include "testFMCMGlobalStore.h"
// INPUT fields:
// OUTPUT fields:


using namespace testFMCM;

UTCcommand::UTCcommand(const string& name,
 AbstractRTAction::RTActionConfig& rtActCfg) :
RTAction<RTEvent, testFMCMGlobalStore, testFMCMDevice>(name, rtActCfg){}


/*UTCcommand::execute
 *
 *Function which instantiates the sending and receiving of UTC data
 * */
void UTCcommand::execute(RTEvent * pEv){

// /*TEST*/
// testFMCMDevice * pDev = deviceCollection[0];
// int fd= pDev->fd.get();
// int count;
// char dummyBuffer[32];
// do{
// count = testFMCMRT::Read(fd,dummyBuffer, 32);
// cout<<dummyBuffer<<endl;
// }while(count>0);
```

```
//
// /****************************************/


///* SI ANALYSES */
//
// MultiplexingContext* ctx = pEv->getMultiplexingContext();
// cout<<"***********************************************"<<endl;
// log << "executing RT action: testFMCMUTCcommand"<<endInfo;
// SendUTC();
///**********************************************/


//// /*EMC TEST*/
// cout<<"*****************TEST EMC*****************************"<<endl;
//    log << "executing RT action: testFMCMUTCcommand"<<endInfo;
//   MultiplexingContext* ctx = pEv->getMultiplexingContext();
// testFMCMDevice * pDev = deviceCollection[0];
//   PMtrigger(pDev, ctx);
//
//
/****************************************/

/*  TEST FMCMS*/
// MultiplexingContext* ctx = pEv->getMultiplexingContext();
// cout<<"***********************************************"<<endl;
// log << "executing RT action: testFMCMUTCcommand"<<endInfo;
// SendUTC();
// testFMCMRT::Sleep(0,60);
// ReceiveUTC(ctx);

/*******************************************************/
MultiplexingContext* ctx = pEv->getMultiplexingContext();
cout<<"***********************************************"<<endl;
log << "executing RT action: testFMCMUTCcommand"<<endInfo;
int UTCcounter = 0;
testFMCMDevice * pDev = deviceCollection[0];
SendUTC();
testFMCMRT::Sleep(0,60);
while (ReceiveUTC(ctx) == 1 && UTCcounter<5){
SendUTC();
testFMCMRT::Sleep(0,60);
UTCcounter++;

}
pDev->UTCRetransmitted.set(pDev->UTCRetransmitted.get() + UTCcounter);
if(UTCcounter>=5){
cout<<"testFMCMUTCcommand failed, UTC message lost,tried 5 times. SEND ALARM"<<endl;
pDev->UTCFailed.set(pDev->UTCFailed.get() + 1);
```

```
}

}

/*UTCcommand::SendUTC
 *
 *Function which writes the UTC command to the RS-422 interface and the FMCM
 * */
void UTCcommand::SendUTC(){

int fd = -1;
int sentcnt = 0;
//for (unsigned int i=0; i < deviceCollection.size(); i++){
testFMCMDevice * pDev = deviceCollection[0];
fd = pDev->fd.get();
    SetUTCCommandData();
    int SentDataSize= strlen(UTCCommandData);
sentcnt= testFMCMRT::Write(fd,UTCCommandData);
testFMCMRT::Flush(fd);
if(sentcnt!= SentDataSize ){
  if(sentcnt==-1){
   cout<<pDev->name.get()<<"(fd"<<fd<<"):
   writing UTC Command request failed with:" ; cout.flush();
  }
}
else{
cout<< "Sending UTC Command to "<<pDev->name.get()
<<" , fd = "<<fd<<" UTC Command: "<<endl;
cout<<UTCCommandData<<"   Number of bytes sent:"<<sentcnt<<endl;
pDev->UTCSent.set(pDev->UTCSent.get() +1);
}
// /* setting data in the Fesa server*/
// for(unsigned int i=0;i<10;i++){
// pDev->SynchronizationCommand.setCell(i,(short)UTCCommandData[i]);
// }
// pDev->StartOfCommand.set(UTCCommandData[10]);
// pDev->CommandCode.set((short)UTCCommandData[11]);
// for(unsigned int i=0; i< 6; i++ ){
// pDev->CommandArg.setCell(i, (short)UTCCommandData[i+12]);
// }
// for(unsigned int i=0;i<2;i++){
// pDev->CommandChecksum.setCell(i, (short)UTCCommandData[i+18]);
// }
log << "UTCcommand operates on device "
<< pDev->name.get()
<< endInfo;
//}
}
```

```
/*UTCcommand::SetUTCCommandData
 *
 *Function which builds the UTC command in a vector called UTCCommandData
 * */
void UTCcommand::SetUTCCommandData(){
long long nanoseconds = 0;
long seconds = 0;
short checksum= 0;
short lsb, msb;
short UTC1, UTC2, UTC3, UTC4;
static string timestring;
/*Calculate UTC time in seconds from nanoseconds and represent the result in four bytes*/
nanoseconds = Timing::getActualTime();
seconds = nanoseconds/(long) 1000000000;
UTC4 = seconds & 0xFF;
UTC3 = (seconds>>8) & 0xFF;
UTC2 =  (seconds>>16) & 0xFF;
UTC1 =  (seconds>>24) & 0xFF;
timestring= Timing::getTimeString(nanoseconds, "%Y/%m/%d/ %T");
/*****************************************************************/
for (unsigned int i=0; i<10; i++){
UTCCommandData[i]= 0x0D ; // Synchronization
}
UTCCommandData[10]= 0x2A; // Start of command
UTCCommandData[11]= 0x74; // UTC command t
UTCCommandData[12]= UTC4; //
UTCCommandData[13]= UTC3;
UTCCommandData[14]= UTC2;
UTCCommandData[15]= UTC1;
UTCCommandData[16]= 0x30;  // 0
UTCCommandData[17]= 0x30;  // 0
for(unsigned int i=11;i<18;i++){
checksum = checksum + (int)UTCCommandData[i];
}
checksum = checksum + 0x55aa;
    lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF ;
UTCCommandData[18]= msb;
UTCCommandData[19]= lsb;
UTCCommandData[20]= '\0';
}



/*UTCcommand::ReceiveUTC
 *
 *Function which reads the UTC data set
```

```
 from the RS-422 interface and the FMCM.
 *It reads first the Response Header,
  then the checksum and the trailer, and
 *in the end returns 0 if checksum was correct.
 *subfunctions:  ReceiveTrailerCheck(), ResponseChksum()
 *
 *Return 0 = success, Return 1 = error
 * */
int UTCcommand::ReceiveUTC(MultiplexingContext* ctx){

int buffersize = 28; // Expected size of Receive Header is 28 bytes
char rcvbuf[buffersize];
int fd = -1;
int rcvcnt= 0;
for (unsigned int i=0; i < deviceCollection.size(); i++){
testFMCMDevice * pDev = deviceCollection[i];
fd= pDev->fd.get();
    /* Read the Response Header:*/
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
if(rcvcnt< buffersize){
cout<<"Error reading Response Header: rcvcnt: "<< rcvcnt<< "retransmit."<<endl;
if(rcvcnt==0){
cout<<"Error reading Response Header: End of line"<<endl;
}
testFMCMRT::Flush(fd);
return 1;
}
if(rcvcnt==buffersize){
// for (unsigned i=0; i<28; i++){
// cout<< (i+1) << ": "<< rcvbuf[i]<<endl;
// }
// cout<<"Reading Response Header"<<endl;
pDev->SynchronizationResponse.set(rcvbuf[0]);
pDev->StartOfCommandReturned.set(rcvbuf[1]);
pDev->CommandCodeReturned.set(rcvbuf[2]);
for(unsigned int i=0; i< 6; i++ ){
pDev->CommandArgReturned.setCell(i, rcvbuf[i+3]);
}
for(unsigned int i=0;i<2;i++){
pDev->CommandChecksumReturned.setCell(i, rcvbuf[i+9]);
}
  for(unsigned int i=0;i<8;i++){
  short temp = rcvbuf[11];
pDev->Errorbits.setCell(i, getbit(7-i,temp));
}
for(unsigned int i=0;i<7;i++){
pDev->CurrentTimestamp.setCell(i, rcvbuf[i+12]);
}
for(unsigned int i=0;i<8;i++){
```

```
short temp = rcvbuf[19];
pDev->InfoBits.setCell(i, getbit(7-i,temp));
}
/*ENDURANCE RUN */
for(unsigned int i=0;i<7;i++){
pDev->TimestampLastPM.setCell(i, rcvbuf[i+12]);
}

/*********************************************/
// for(unsigned int i=0;i<7;i++){
// pDev->TimestampLastPM.setCell(i, rcvbuf[i+20]);
// }
if(rcvbuf[27]==0x00){
pDev->Spare.set(0x30);
}
else{
pDev->Spare.set(rcvbuf[27]);
}
/*Check the type of command being sent and set the fields  */
if (rcvbuf[2] == 't'){
if(ReceiveTrailerCheck(fd,pDev)== 0){
// if (ResponseChksum(rcvbuf,pDev,buffersize)== true){
// cout<<rcvbuf[2]<<" Response received"<<endl;
// }
// else{
// cout<<"Response type: "<<rcvbuf[2] <<" ResponseChecksum inncorrect, retransmit ?"<<end
// testFMCMRT::Flush(fd);
// return 1;
//
// }
}
else{
cout<<"Error while receiving trailer and checksum, retransmit."<<endl;
testFMCMRT::Flush(fd);
return 1;

}
}
else{
cout<<"Unknown response command, retransmit."<<endl;
testFMCMRT::Flush(fd);
return 1;
}
}

CheckTime(pDev);
cout<<"update UTC message received"<<endl;
pDev->UTCReceived.set(pDev->UTCReceived.get() +1);
PMtrigger(pDev,ctx);
```

```
log << "UTCcommand operates on device "
<< pDev->name.get()
<< endInfo;
testFMCMRT::Flush(fd);
}
return 0;
}

/*Function getbit
 *
 * Get bit number p from a char(8 bits), counting from the right.
 * 0=<p>8
 * ************************************************************************/
int UTCcommand::getbit(int p, char x){

return((x >> (p)) & ~(~0 << 1));
}


/*UTCcommand::ReceiveTrailerCheck
 *
 * Function to read the remaining parts of the status Message from the FMCM
 * Assumes Receive Header has already been read
 * Response Checksum: 2 bytes, Trailer: 2 bytes
 *
 * Return 0 = success, Return 1 = error
 * */
int UTCcommand::ReceiveTrailerCheck(int fd, testFMCMDevice* pDev){

int buffersize = 4; // Expected size is 4 bytes
char rcvbuf[buffersize];
int rcvcnt=0;
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
    if(rcvcnt< buffersize){
cout<<"Error reading Trailer and Checksum: rcvcnt<buffersize"<<rcvcnt<<endl;
if(rcvcnt==0){
cout<<"Error reading Trailer and Checksum: End of line"<<endl;
}
return 1;
}
if(rcvcnt==buffersize){
// for (unsigned i=0; i<4; i++){
// cout<< "trailcheck"<<(i) << ": "<< rcvbuf[i]<<endl;
// }
for(unsigned int i=0;i<2;i++){
pDev->ResponseChecksum.setCell(i, rcvbuf[i+0]);
}
for(unsigned int i=0;i<2;i++){
pDev->ResponseTrailer.setCell(i, rcvbuf[i+2]);
```

```
}
}
return 0;
}

/*StatusCommand::ResponseChksum
 *
 *Function which calculates the Checksum of the entire PM command.
 *Returns true if calculated checksum is identical with received checksum.
 * */
bool UTCcommand::ResponseChksum(char * responseHeader,
testFMCMDevice * pDev,int buffersize){

unsigned int size = buffersize;
short checksum =0;
short lsb, msb = 0;
for (unsigned int i= 0; i<size;i++){   //
checksum = checksum + responseHeader[i];
}
checksum = checksum + 0x55aa;
lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF;

cout<< "msb: "<<msb<<"  lsb: "<<lsb<<endl;
if ((msb == pDev->ResponseChecksum.getCell(0))
&& ( lsb== pDev->ResponseChecksum.getCell(1))){
  return true;
}
else {
return false;
}
}

/*StatusCommand::NotifyPropertyPM
 *
 * */
void UTCcommand::NotifyPropertyPM(testFMCMDevice * pDev,
MultiplexingContext* ctx){

std:: string className = pGlobalStore->name.get();
std:: string notificationstr1;
notificationstr1 = "PmData:";
notificationstr1 += pDev->name.get();
notificationstr1 += ",";
AbstractEquipmentRT::notify(ctx, className, notificationstr1);
std:: string notificationstr2;
notificationstr2 = "Emc:";
notificationstr2 += pDev->name.get();
notificationstr2 += ",";
```

```
AbstractEquipmentRT::notify(ctx, className, notificationstr2);
}


/*UTCcommand::PMtrigger(
 *
 *If timestamp of last PM aquisition has canged,
 * get PM data, update the PMTimestamp field
 *and notify the property PMdata
 * */
 void  UTCcommand::PMtrigger(testFMCMDevice * pDev,MultiplexingContext* ctx){

//  /*TEST EMC*/
//
// PMcommand::GetPMData(pDev);
// NotifyPropertyPM(pDev, ctx);
//
// /*************************************************************/


  /*ENDURANCE RUN*/
  if (pDev->TimestampLastPM.getCell(2)!=pDev->PMTimestamp.getCell(2)){
PMcommand::GetPMData(pDev);
for(unsigned int i = 0; i<7; i++){
pDev->PMTimestamp.setCell(i,pDev->TimestampLastPM.getCell(i));
}
NotifyPropertyPM(pDev, ctx);
}

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

// if (pDev->TimestampLastPM.getCell(0)!=pDev->PMTimestamp.getCell(0) ||
// pDev->TimestampLastPM.getCell(1)!=pDev->PMTimestamp.getCell(1) ||
// pDev->TimestampLastPM.getCell(2)!=pDev->PMTimestamp.getCell(2) ||
// pDev->TimestampLastPM.getCell(3)!=pDev->PMTimestamp.getCell(3) ||
// pDev->TimestampLastPM.getCell(4)!=pDev->PMTimestamp.getCell(4) ||
// pDev->TimestampLastPM.getCell(5)!=pDev->PMTimestamp.getCell(5) ||
// pDev->TimestampLastPM.getCell(6)!=pDev->PMTimestamp.getCell(6)){
//
//
// PMcommand::GetPMData(pDev);
// for(unsigned int i = 0; i<7; i++){
// pDev->PMTimestamp.setCell(i,pDev->TimestampLastPM.getCell(i));
// }
// NotifyPropertyPM(pDev, ctx);
// }

 }
```

```
 /*
  * UTCcommand::CheckTime
  *
  * Testfunction to see if the FMCM understood the last UTC update
  * */
void UTCcommand::CheckTime(testFMCMDevice * pDev){
 static string timestring;
 long longUTC4 = (long)pDev->CurrentTimestamp.getCell(3);
 long longUTC3 = ((long)pDev->CurrentTimestamp.getCell(2))<<8;
 long longUTC2 = ((long)pDev->CurrentTimestamp.getCell(1))<<16;
 long longUTC1 = ((long)pDev->CurrentTimestamp.getCell(0))<<24;

 long seconds = longUTC4+ longUTC3+ longUTC2+ longUTC1;
 long long nseconds = seconds*(long long)1000000000 ;
 timestring= Timing::getTimeString(nseconds, "%Y/%m/%d/ %T");
// cout<<"Current UTC time:"<<timestring<<endl;
 "<<pDev->CurrentTimestamp.getCell(2)<< " UTC4: "<< pDev->CurrentTimestamp.getCell(3)<<en
}
```

Chapter 7

SetCommand.h

```
//
//  FESA framework June 2004.
//  version 1
// Use this code as a starting-point to develop your own equipment class

#ifndef _testFMCM_SetCommand_H_
#define _testFMCM_SetCommand_H_

#include <string>
#include <fesa/Fesa.h>
#include "testFMCMTypeDefinition.h"

class RequestEvent;
#include "testFMCMDevice.h"
#include "testFMCMGlobalStore.h"

namespace testFMCM {

class SetCommand: public ServerAction <testFMCMGlobalStore,
 testFMCMDevice, SetCommand_DataType> {
public:
void execute(RequestEvent *);
SetCommand(const string& name,
AbstractServerAction::ServerActionConfig& serverActCfg ) ;

};
}
#endif
```

Chapter 8

SetCommand.cpp

```
//
//  FESA framework June 2004.
//  version 1
// Use this code as a starting-point to develop your own equipment class

#include <SetCommand.h>
#include <testFMCMDevice.h>
#include <testFMCMGlobalStore.h>
#include <testFMCMInterface.h>
// INPUT fields:
// OUTPUT fields:


using namespace testFMCM;

SetCommand::SetCommand(const string& name,
AbstractServerAction::ServerActionConfig& serverActCfg) :
ServerAction<testFMCMGlobalStore, testFMCMDevice,
 SetCommand_DataType >(name, serverActCfg){}

void SetCommand::execute(RequestEvent * pEv){
cout<<"*****************************************"<<endl;
log << "executing server action: testFMCMSetCommand"<<endInfo;
MultiplexingContext* pContext ;
pContext = pEv->getMultiplexingContext();

char os[256];
const char* pDevName = pWorkingDevice->name.get(pContext);
const short* temp;
unsigned long  size = 6;
UserEvent evt= NewCommand;

temp = data.CommandArg.get(size);
for(unsigned int i=0;i<6;i++){
pWorkingDevice->CommandArg.setCell(i, temp[i], pContext);
```

```
}
pWorkingDevice->CommandCode.set(data.CommandCode.get(), pContext);

testFMCMInterface * p2testFMCMInterface = dynamic_cast<testFMCMInterface *>(AbstractEquip
sprintf(os, "%s",  pDevName);
p2testFMCMInterface->fireUserEvent(evt, os);   /
}
```

Chapter 9

SetCommandRT.h

```
//
//  FESA framework June 2004.
//  version1
// Use this code as a starting-point to develop your own equipment class

#ifndef _testFMCM_SetCommandRT_H_
#define _testFMCM_SetCommandRT_H_

#include <fesa/Fesa.h>
#include "testFMCMDevice.h"
#include "testFMCMGlobalStore.h"

namespace testFMCM {

class SetCommandRT: public RTAction <RTEvent, testFMCMGlobalStore, testFMCMDevice >
public:
void execute(RTEvent *);
SetCommandRT(const string& name, AbstractRTAction::RTActionConfig& rtActCfg) ;
private:
void  SetCommandData(testFMCMDevice * pDev);
void  SendCommandData(testFMCMDevice * pDev);
char  CommandData[21];
int  ReceiveIRD(testFMCMDevice * pDev,
MultiplexingContext* ctx );
int  ReceiveTrailerCheck(int fd, testFMCMDevice * pDev);
int  getbit(int p, char x);
bool  ResponseChksum(char * responseHeader,
testFMCMDevice * pDev,int buffersize);
void  NotifyPropertyPM(testFMCMDevice * pDev,
MultiplexingContext* ctx);
void  PMtrigger(testFMCMDevice * pDev,
MultiplexingContext* ctx);
};
}
#endif
```

Chapter 10

SetCommandRT.cpp

```
//
//  FESA framework June 2004.
//  version1
//  Use this code as a starting-point to develop your own equipment class

#include <SetCommandRT.h>
#include <testFMCMRealtime.h>
#include "PMcommand.h"

// INPUT fields:
// OUTPUT fields:


using namespace testFMCM;

SetCommandRT::SetCommandRT(const string& name, A
bstractRTAction::RTActionConfig& rtActCfg) :
RTAction<RTEvent, testFMCMGlobalStore, testFMCMDevice>(name, rtActCfg){}


/*SetCommandRT::execute
 *
 *Function which instaniates the sending and receiving of
 *Idle, Reset Alarm Counter or Simulate Dump data.
 * */
void SetCommandRT::execute(RTEvent * pEv){
MultiplexingContext* ctx = pEv->getMultiplexingContext();
string DevName = pEv->getPayload()->getValue();
testFMCMDevice * pDev = testFMCMDevice::getDevice(DevName);
int SetCommandcounter = 0;
SendCommandData(pDev);
testFMCMRT::Sleep(0,60);

while (ReceiveIRD(pDev,ctx )== 1 && SetCommandcounter<5){
```

```
SendCommandData(pDev);
testFMCMRT::Sleep(0,60);
SetCommandcounter++;
}
pDev->IRDRetransmitted.set(pDev->IRDRetransmitted.get() + SetCommandcounter);
if(SetCommandcounter>=5){
cout<<"SetCommandRT failed, IRD  message lost,tried 5 times. SEND ALARM"<<endl;
pDev->IRDFailed.set(pDev->IRDFailed.get() + 1);
}
log << "SetCommandRT operates on device "
<< pDev->name.get()
<< endInfo;
}

/*SetCommandRT::SendCommandData
 *
 *Function which writes the Idle,
 Reset Alarm Counter or Simulate Dump command
 to the RS-422 interface and the FMCM
 * */
void SetCommandRT::SendCommandData(testFMCMDevice* pDev){
int fd = -1;
int sentcnt = 0;
int SentDataSize=0;
fd = pDev->fd.get();
    SetCommandData(pDev);
    SentDataSize=strlen(CommandData);
    testFMCMRT::Flush(fd);
sentcnt= testFMCMRT::Write(fd,CommandData);
if(sentcnt!= SentDataSize ){
if(sentcnt<=SentDataSize && sentcnt!=0){
cout<<pDev->name.get()<<"(fd"<<fd<<"):
writing Command request. sentcnt:"<<sentcnt <<endl;
}
if(sentcnt==0){
  cout<<pDev->name.get()<<"(fd"<<fd<<"):
  writing Command request. No data transmitted:"<<sentcnt <<endl;
}
}
else{
cout<< "Sending Command to "<<
pDev->name.get()<<" , fd = "<<fd<<"    Command: "<<endl;
cout<<CommandData<<"   Number of bytes sent:"<<sentcnt<<endl;
pDev->IRDSent.set(pDev->IRDSent.get() + 1);
}
/* setting data in the Fesa database*/
for(unsigned int i=0;i<10;i++){
pDev->SynchronizationCommand.setCell(i,(short)CommandData[i]);
}
```

```
pDev->StartOfCommand.set((short)CommandData[10]);
for(unsigned int i=0;i<2;i++){
pDev->CommandChecksum.setCell(i, (short) CommandData[i+18]);
}
}


/*SetCommandRT::SetCommandData
 *
 *Function which builds the command in a vector called CommandData
 * */
void SetCommandRT::SetCommandData(testFMCMDevice * pDev){

int checksum= 0;
short lsb, msb;
for (unsigned int i=0; i<10; i++){
CommandData[i]= 0x0D ; // Synchronization
}
CommandData[10]= 0x2A; //Start of command
CommandData[11]= pDev->CommandCode.get();  //Idle command i
for(unsigned int i=0; i< 6; i++ ){          //Command arguments:
CommandData[12+i]= pDev->CommandArg.getCell(i);
}
for(unsigned int i=11;i<18;i++){
checksum = checksum + (int)CommandData[i];
}
checksum = checksum + 0x55aa;
    lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF ;
CommandData[18]= msb;
CommandData[19]= lsb;
CommandData[20]= '\0';
}

/*SetCommandRT::ReceiveIRD
 *
 *Function which reads the Idle, RAC or
 Dump response from the RS-422 interface and the FMCM.
 *It reads first the Response Header then the
 checksum and the trailer, and
 *in the end returns 0 if checksum was correct.
 *subfunctions:  ReceiveTrailerCheck(), ResponseChksum()
 *
 *Return 0 = success, Return 1 = error
 * */
int SetCommandRT::ReceiveIRD(testFMCMDevice * pDev,MultiplexingContext* ctx ){

int buffersize = 28;
char rcvbuf[buffersize];
```

```
int fd = -1;
int rcvcnt= 0;
fd= pDev->fd.get();
 /* Read the Response Header:*/
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
if(rcvcnt< buffersize){
cout<<"Error reading Response Header: rcvcnt<buffersize"<< rcvcnt<< endl;
if(rcvcnt==0){
cout<<"Error reading Response Header: End of line"<<endl;
}
testFMCMRT::Flush(fd);
return 1;
}
if(rcvcnt==buffersize){
// for (unsigned i=0; i<28; i++){
// cout<< (i+1) << ": "<< rcvbuf[i]<<endl;
// }
/*Sets the different fields in the Receive Header if read was sucessful:*/
cout<<"Reading Response Header"<<endl;
pDev->SynchronizationResponse.set(rcvbuf[0]);
pDev->StartOfCommandReturned.set(rcvbuf[1]);
pDev->CommandCodeReturned.set(rcvbuf[2]);
for(unsigned int i=0; i< 6; i++ ){
pDev->CommandArgReturned.setCell(i, rcvbuf[i+3]);
}
for(unsigned int i=0;i<2;i++){
pDev->CommandChecksumReturned.setCell(i, rcvbuf[i+9]);
}
  for(unsigned int i=0;i<8;i++){
  short temp = rcvbuf[11];
pDev->Errorbits.setCell(i, getbit(7-i,temp));
}
for(unsigned int i=0;i<7;i++){
pDev->CurrentTimestamp.setCell(i, rcvbuf[i+12]);
}
for(unsigned int i=0;i<8;i++){
short temp = rcvbuf[11];
pDev->InfoBits.setCell(i, getbit(7-i,temp));
}
/*ENDURANCE RUN */
for(unsigned int i=0;i<7;i++){
pDev->TimestampLastPM.setCell(i, rcvbuf[i+12]);
}

/*********************************************/
// for(unsigned int i=0;i<7;i++){
// pDev->TimestampLastPM.setCell(i, rcvbuf[i+20]);
// }
if(rcvbuf[27]==0x00){
```

```
pDev->Spare.set(0x30);
}
else{
pDev->Spare.set(rcvbuf[27]);
}
/*Choose the type of command being sent
 read the rest, set the fields and update the property: */
if (rcvbuf[2] == 'i'|| rcvbuf[2] == 'r'|| rcvbuf[2] == 'd'){
if(ReceiveTrailerCheck(fd,pDev)== 0){
// if (ResponseChksum(rcvbuf,pDev,buffersize)== true){
// cout<<rcvbuf[2]<<" Response received"<<endl;
// }
// else{
// return 1;
// }
}
else{
return 1;
}
}
else{
return 1;
}
testFMCMRT::Flush(fd);
}
cout<<rcvbuf[2] <<" message received"<<endl;
pDev->IRDReceived.set(pDev->IRDReceived.get() + 1);
PMtrigger(pDev,ctx);

return 0;
}

/*Function getbit
 *
 * Get bit number p from a char(8 bits), counting from the right.
 * 0=<p>8
 * ************************************************************************/
int SetCommandRT::getbit(int p, char x){

return((x >> (p)) & ~(~0 << 1));
}


/*Function to read the remaining parts of the
* i, t, r or d Message from the FMCM
 * Assumes Receive Header has already been read
 * Data:0 bytes, Response Checksum: 2 bytes, Trailer: 2 bytes
 *
 * */
```

```
int SetCommandRT::ReceiveTrailerCheck(int fd, testFMCMDevice* pDev){
int buffersize = 4; // Expected size is 4 bytes
char rcvbuf[buffersize];
int rcvcnt=0;
rcvcnt=testFMCMRT::Read(fd, rcvbuf,buffersize);
    if(rcvcnt< buffersize){
cout<<"Error reading Trailer and Checksum: rcvcnt<buffersize"<<rcvcnt<<endl;
if(rcvcnt==0){
cout<<"Error reading Trailer and Checksum: End of line"<<endl;
}
return 1;
}
if(rcvcnt==buffersize){
// for (unsigned i=0; i<4; i++){
// cout<< "trailcheck"<<(i) << ": "<< rcvbuf[i]<<endl;
// }
for(unsigned int i=0;i<2;i++){
pDev->ResponseChecksum.setCell(i, rcvbuf[i+0]);
}
for(unsigned int i=0;i<2;i++){
pDev->ResponseTrailer.setCell(i, rcvbuf[i+2]);
}
}
return 0;
}


/*SetCommandRT::ResponseChksum
 *
 *Function which calculates the Checksum
 *of the entire PM command.
 *Returns true if calculated checksum is
 *identical with received checksum.
 * */
bool SetCommandRT::ResponseChksum(char * responseHeader,
testFMCMDevice * pDev,int buffersize){

unsigned int size = buffersize;
short checksum =0;
short lsb, msb = 0;
for (unsigned int i= 1; i<size;i++){
checksum = checksum + responseHeader[i];
// cout<< "check"<< i<< ": "<< checksum<<endl;
}
checksum = checksum + 0x55aa;
lsb = checksum & 0xFF;
msb = (checksum>>8 ) & 0xFF;
if ((msb == pDev->ResponseChecksum.getCell(0)) &&
( lsb== pDev->ResponseChecksum.getCell(1))){
```

```
  return true;
}
else {
return false;
}
}


/*SetCommandRT::NotifyPropertyPM
 *
 * */
void SetCommandRT::NotifyPropertyPM(testFMCMDevice * pDev,
MultiplexingContext* ctx){

std:: string className = pGlobalStore->name.get();
std:: string notificationstr;
notificationstr = "PmData:";
notificationstr += pDev->name.get();
AbstractEquipmentRT::notify(ctx,className, notificationstr);

}


/*SetCommandRT::PMtrigger(
 *
 *If timestamp of last PM aquisition has canged,
 * get PM data, update the PMTimestamp field
 *and notify the property PMdata
 * */
 void  SetCommandRT::PMtrigger(testFMCMDevice * pDev,
 MultiplexingContext* ctx){

  /*ENDURANCE RUN*/
  if (pDev->TimestampLastPM.getCell(2)!=pDev->PMTimestamp.getCell(2)){
PMcommand::GetPMData(pDev);
for(unsigned int i = 0; i<7; i++){
pDev->PMTimestamp.setCell(i,pDev->TimestampLastPM.getCell(i));
}
NotifyPropertyPM(pDev, ctx);
}

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

/* if (pDev->TimestampLastPM.getCell(0)!=pDev->PMTimestamp.getCell(0) ||
pDev->TimestampLastPM.getCell(1)!=pDev->PMTimestamp.getCell(1) ||
pDev->TimestampLastPM.getCell(2)!=pDev->PMTimestamp.getCell(2) ||
pDev->TimestampLastPM.getCell(3)!=pDev->PMTimestamp.getCell(3) ||
pDev->TimestampLastPM.getCell(4)!=pDev->PMTimestamp.getCell(4) ||
pDev->TimestampLastPM.getCell(5)!=pDev->PMTimestamp.getCell(5) ||
```

```
pDev->TimestampLastPM.getCell(6)!=pDev->PMTimestamp.getCell(6)){
PMcommand::GetPMData(pDev);
for(unsigned int i = 0; i<7; i++){
pDev->PMTimestamp.setCell(i,pDev->TimestampLastPM.getCell(i));
}
NotifyPropertyPM(pDev, ctx);
} */
 }
```

Chapter 11

testFMCMRT.h

```
// This class is not mandatory : developper
// have to provide it only if he wants to override
// specificInit to execute some specific action when
// RT part of the equipment software is initialized
// at the start-up.

#ifndef _testFMCM_REALTIME_H_
#define _testFMCM_REALTIME_H_

#include <string>

#include <fesa/Fesa.h>
#include "testFMCMEquipmentDefaultRealtime.h"
#include "testFMCMDevice.h"

namespace testFMCM {

class testFMCMRT : public testFMCMEquipmentDefaultRT {
public:
testFMCMRT(const string& name, const string& descPath,
AbstractEquipmentClass::FesaDeployType deployType);
virtual void specificInit(int argc, char ** argv);
virtual ~testFMCMRT();
static void Connect(testFMCMDevice* dev);
static int  GetFileDescriptor(const char* dn, short mezzanine,
 short motherboard, short channel);
static int  Read(int fd,char* message, int maxlen);
static int  Write(int fd,char* message);
static void Sleep(long s, long ms);
static void Flush(int fd);
};
}
#endif
```

Chapter 12

testFMCMRT.cpp

```
// This class is not mandatory : developper
// have to provide it only if he wants to override
// specificInit to execute some specific action when
// RT part of the equipment software is initialized
// at the start-up.


#include "testFMCMRealtime.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <unistd.h>
#include <stropts.h>
#include <termio.h>
#include <file.h>
#include <sys/types.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <gm/moduletypes.h>


extern "C"{ int ip8GetLogicalLineFileName ();};


using namespace testFMCM;



testFMCMRT::testFMCMRT(const string& name, const string& descPath,
AbstractEquipmentClass::FesaDeployType deployType) :
testFMCMEquipmentDefaultRT(name, descPath, deployType) {
}
```

```
testFMCMRT::~testFMCMRT() {
}

/*testFMCMRT::specificInit
 *
 *Initilizes the RS-422 communication
 ******************************/
void testFMCMRT::specificInit(int argc, char ** argv) {
cout << " testFMCMRT::specificInit is called" << endl;
vector<testFMCMDevice*>* devices  = getDeviceCollection();
//unsigned int dev_cnt = devices->size();
testFMCMDevice* dev = (*devices)[0];
Connect(dev);
}


/*testFMCMRT::GetFileDescriptor
 *
 * Returns filesdescriptor fd
 * fd -1 = error
 * ******************************/
int testFMCMRT::GetFileDescriptor(const char* dn,
short mezzanine, short motherboard, short channel){

struct termio s;
int fd = -1;
int c= 0;
char filename[256];
c = ip8GetLogicalLineFileName(mezzanine, motherboard,channel, &filename);
if (c){
            cout<<"ip8GetLogicalLineFileName: Error.
             Not able to compute file handle for lun:
             " <<motherboard<<", channel: "<<channel<<endl;
          return -1;
        }
else{
cout<<"ip8GetLogicalLineFileName: computed file handle for
 lun:"<<motherboard<<", channel:"<<channel<<" is :"<<filename << endl;
}
   fd = open(filename,O_RDWR | O_NDELAY);
//  if( (fd==-1) || (ioctl(fd,TCGETA,&s)==-1) ) {//get settings termio
//     cout<<"ioctl(Get settings) error, FD------>"<<fd <<endl;
//      return -1;
//  }
s.c_iflag = 0; //INPCK | IXON |IXOFF;
s.c_oflag= 0;
s.c_cflag=   CS8|V_B38400|PARENB|PARODD|CREAD|CLOCAL ;
s.c_lflag=0;
s.c_line= 0;
```

```
s.c_cc[VMIN] = 0;
s.c_cc[VTIME]  = 0;

  if (ioctl(fd, TCSETA, &s) == -1){
        cout << "ioctl(Set settings) error,  FD-->" << fd << endl;
        return(-1);
     }
return fd;
}


/*testFMCMRT::Connect
 *
 *
 **************************** */
void testFMCMRT::Connect(testFMCMDevice* dev){
  const char* devname     = dev->name.get();
  short motherboardnumber = dev->hw1Lun.get();
  short mezzaninetype  = IocVIP626IP8R422;
  short  channelnumber    = dev->hw1Ch.get();
  int fd = GetFileDescriptor(devname,mezzaninetype, motherboardnumber,channelnumber)

  if(fd>0){
    cout<<dev->name.get()<<"(fd "<<fd<<") responding"<<endl;
    dev->fd.set(fd);
  }
  else{
    cout<<dev->name.get()<<"(fd "<<fd<<") not responding"<<endl;
    dev->fd.set(-1);
  }


}

/*testFMCMRT::Read
 *
 **************************** */
int testFMCMRT::Read(int fd,char* message, int maxlen){
  return read(fd,message,maxlen);
}

/*testFMCMRT::Write
 *
 **************************** */
int testFMCMRT::Write(int fd,char* message){
  return write(fd,message,strlen(message));
}

/*testFMCMRT::Sleep
```

```
 *
 ************************/
void testFMCMRT::Sleep(long s, long ms){
struct timespec t= {(long)s, (time_t)((time_t)ms*(time_t)100000)};
if(nanosleep(&t, &t)){
switch(errno){
case EINTR:
cout<<"nanosleep failed (EINTR)"<<endl;
break;
case EINVAL:
cout<<"nanosleep failed (EINVAL)"<<endl;
break;
case EAGAIN:
cout<<"nanosleep failed (EAGAIN)"<<endl;
break;
default:
cout<<"nanosleep failed - errno = "<<errno<<endl;
break;
};
nanosleep (&t, &t);
};
}

/*testFMCMRT::Flush
 *
 * Clean buffer
 ********************************** */
void testFMCMRT::Flush(int fd){

int count;
char dummyBuffer[512];
do{
count = Read(fd,dummyBuffer, 512);
}while(count>0);

}
```