# CFM and Management Entity

## Stop the module

**SYNOPSIS**

**int cfmSupervisory\_Stop(cfm\_t cfm);**

**TestCase**

|  |
| --- |
| ret = ioctl(fd, CFM\_STOP);  if(ret != 0){  printf("ioctl error\n");  } |

## Create MD

**SYNOPSIS**

**int cfmSupervisory\_CreateMD(uint16 \*meid, uint8 MDLevel, uint8 MDnameFormat, uint8 MDnameLength, uint8 \*MDname);**

**TestCase**

|  |
| --- |
| struct Test\_MD\_st Test\_MD;  memset(&Test\_MD, 0 ,sizeof(struct Test\_MD\_st ));  Test\_MD.MDLevel = 4;  Test\_MD.MDnameFormat = 1;  Test\_MD.MDnameLength = 20;  memcpy(Test\_MD.MDname,"the new MD" ,20);  ret = ioctl(fd, CFM\_CREATE\_MD ,&Test\_MD);  if(ret != 0){  printf("ioctl error\n");  }  else printf("new MD created!MEID:%d\n",Test\_MD.meid); |

## Delete MD

**SYNOPSIS**

**int cfmSupervisory\_DeleteMD(uint16 meid);**

**TestCase**

|  |
| --- |
| struct Test\_MD\_st Test\_MD;  memset(&Test\_MD, 0 ,sizeof(struct Test\_MD\_st ));  Test\_MD.meid =1;  ret = ioctl(fd, CFM\_DELETE\_MD,&Test\_MD);  if(ret != 0){  printf("ioctl error\n");  } |

## Get MD

**SYNOPSIS**

**int cfmSupervisory\_GetMD(void \*result);**

**TestCase**

|  |
| --- |
| struct Test\_MD\_st Test\_MD;  memset(&Test\_MD, 0 ,sizeof(struct Test\_MD\_st ));  Test\_MD.meid = 1;  ret = ioctl(fd, CFM\_GET\_MD,&Test\_MD);  if(ret != 0){  printf("ioctl error\n");  }  else printf("Level:%d,NameFormat:%d,MHFCreation:%d\n",Test\_MD.MDLevel,Test\_MD.MDnameFormat,Test\_MD.MHFCreation); |

## Set MD

**SYNOPSIS**

**int cfmSupervisory\_SetMD(uint16 meid, uint8 event, uint8 value, uint8 \*MDname);**

**TestCase**

|  |
| --- |
| struct Test\_MD\_st Test\_MD;  memset(&Test\_MD, 0 ,sizeof(struct Test\_MD\_st ));  Test\_MD.meid = 1;  Test\_MD.event=3;//Level  Test\_MD.value=6;  ret = ioctl(fd, CFM\_SET\_MD,&Test\_MD);  if(ret != 0){  printf("ioctl error\n");  } |

## Create MA

**SYNOPSIS**

**int cfmSupervisory\_CreateMA(uint16 \*meid, uint16 MDId, uint8 ShortMAFormat, uint8 ShortMAnameLength, uint8 \*ShortMAname, uint16 \*AssociatedVLANs);**

**TestCase**

|  |
| --- |
| struct Test\_MA\_st Test\_MA;  memset(&Test\_MA, 0 ,sizeof(struct Test\_MA\_st));  Test\_MA.MDid = 1;  Test\_MA.ShortMAnameFormat = 0x01;  Test\_MA.ShortMAnameLength = 20;  memcpy( Test\_MA.ShortMAname, "the new MA ",20);  ret = ioctl(fd, CFM\_CREATE\_MA,&Test\_MA);  if(ret != 0){  printf("ioctl error\n");  } |

## Delete MA

**SYNOPSIS**

**int cfmSupervisory\_DeleteMA(uint16 meid);**

**TestCase**

|  |
| --- |
| struct Test\_MA\_st Test\_MA;  memset(&Test\_MA, 0 ,sizeof(struct Test\_MA\_st));  Test\_MA.meid=1;  ret = ioctl(fd, CFM\_DELETE\_MA,&Test\_MA);  if(ret != 0){  printf("ioctl error\n");  } |

## Get MA

**SYNOPSIS**

**int cfmSupervisory\_GetMA(void \*result);**

**TestCase**

|  |
| --- |
| struct Test\_MA\_st Test\_MA;  memset(&Test\_MA, 0 ,sizeof(struct Test\_MA\_st));  Test\_MA.meid= 1;  ret = ioctl(fd, CFM\_GET\_MA,&Test\_MA);  if(ret != 0){  printf("ioctl error\n");  }  else  printf("MDID:%d,MAName:%s\n",Test\_MA.MDid,Test\_MA.ShortMAname); |

## Set MA

**SYNOPSIS**

**int cfmSupervisory\_SetMA(uint16 meid, uint8 event, uint8 value, uint16 mdid, void \*arg);**

**TestCase**

|  |
| --- |
| struct Test\_MA\_st Test\_MA;  memset(&Test\_MA, 0 ,sizeof(struct Test\_MA\_st));  Test\_MA.meid=1;  Test\_MA.event=2;  Test\_MA.value=0x02;  ret = ioctl(fd, CFM\_SET\_MA,&Test\_MA);  if(ret != 0){  printf("ioctl error\n");  } |

## Create MEP

**SYNOPSIS**

**int cfmSupervisory\_CreateMEP(uint16 \*meid, uint16 MAid, uint16 MEPId, uint8 Direction, Layer2\_Entity\_t layer2Entity, uint16 FlowID);**

**TestCase**

|  |
| --- |
| struct Test\_MEP\_Create\_st Test\_MEP\_Create;  uint8 TMAC[ADDR\_LEN]={0x00,0x0c,0x29,0xed,0xfc,0xdd};  memset(&Test\_MEP\_Create,0,sizeof(struct Test\_MEP\_Create\_st));  Test\_MEP\_Create.Direction=0x02;  Test\_MEP\_Create.FlowID=2;  Test\_MEP\_Create.srcPortId=1;  Test\_MEP\_Create.MAid=2;  memcpy(Test\_MEP\_Create.MACAddr,TMAC,ADDR\_LEN);  ret=ioctl(fd,CFM\_CREATE\_MEP,&Test\_MEP\_Create);  if(ret!=0)  {  printf("ioctl error\n");  }  else  printf("Create MEPID=%d\n",Test\_MEP\_Create.meid); |

## Delete MEP

**SYNOPSIS**

**int cfmSupervisory\_DeleteMEP(uint16 meid);**

**TestCase**

|  |
| --- |
| uint16 meid=2;  ret=ioctl(fd,CFM\_DELETE\_MEP,&meid);  if(ret!=0)  {  printf("ioctl error\n");  } |

## Get MEP

**SYNOPSIS**

**int cfmSupervisory\_GetMEP(uint16 meid, void \*result);**

**TestCase**

|  |
| --- |
| struct Test\_MEP\_Get\_st Test\_MEP\_Get;  memset(&Test\_MEP\_Get,0,sizeof(struct Test\_MEP\_Get\_st));  Test\_MEP\_Get.meid=1;  ret=ioctl(fd,CFM\_GET\_MEP,&Test\_MEP\_Get);  if(ret != 0){  printf("ioctl error\n");  }  else  printf("FlowID:%d,Direction:%d,VLAN:%d\n",Test\_MEP\_Get.FlowId,Test\_MEP\_Get.Direction,Test\_MEP\_Get.PrimaryVlan); |

## Set MEP

**SYNOPSIS**

**int cfmSupervisory\_SetMEP(uint16 meid, uint8 event, uint8 value8, uint16 value16 );**

**TestCase**

|  |
| --- |
| struct Test\_MEP\_Set\_st Test\_MEP\_Set;  memset(&Test\_MEP\_Set,0,sizeof(struct Test\_MEP\_Set\_st));  Test\_MEP\_Set.meid=1;  Test\_MEP\_Set.event=1;  Test\_MEP\_Set.value16=2;  ret=ioctl(fd,CFM\_SET\_MEP,&Test\_MEP\_Set);  if(ret != 0){  printf("ioctl error\n");  } |

## Get MP Status

**SYNOPSIS**

**int cfmSupervisory\_GetCFMMPStatus(void \*param);**

**TestCase**

|  |
| --- |
| struct Test\_MPStatusEntry\_st MPStatusEntry;  memset(&MPStatusEntry,0,sizeof(struct Test\_MEP\_Get\_st));  MPStatusEntry.Direction=1;  MPStatusEntry.VlanId=20;  MPStatusEntry.Level=5;  MPStatusEntry.PortId=1;  ret=ioctl(fd,CFM\_GET\_MPSTATUS,&MPStatusEntry);  if(ret != 0){  printf("ioctl error\n");  }  else printf("meid:%d\n",MPStatusEntry.MEPId); |

## Get Configuration Error

**SYNOPSIS**

**int cfmSupervisory\_GetCFMConfigurationError(void\*param);**

**TestCase**

|  |
| --- |
| struct Test\_ConfigurationErrorEntry\_st ConfigurationErrorEntry;  memset(&ConfigurationErrorEntry,0,sizeof(struct Test\_ConfigurationErrorEntry\_st));  ret=ioctl(fd,CFM\_GET\_CONFIGERROR,&ConfigurationErrorEntry);  if(ret != 0){  printf("ioctl error\n");  }  else  printf("DetectedConfigurationError=%d\n",ConfigurationErrorEntry.DetectedConfigurationError); |

## Get CFM Stack

**SYNOPSIS**

**int cfmSupervisory\_GetCFMStack(void \*pstack);**

**TestCase**

|  |
| --- |
| struct Test\_CFMStack\_st Test\_CFMStack;  memset(&Test\_CFMStack,0,sizeof(struct Test\_CFMStack\_st));  ret=ioctl(fd,CFM\_GET\_STACK,&Test\_CFMStack);  if(ret != 0){  printf("ioctl error\n");  } |

## Get MDList

**SYNOPSIS**

**int cfmSupervisory\_GetMDList(void \* MDList);**

**TestCase**

|  |
| --- |
| struct Test\_MDList\_st Test\_MDList;  int i;  memset(&Test\_MDList,0,sizeof(Test\_MDList\_st));  ret=ioctl(fd,CFM\_GET\_MDLIST,&Test\_MDList);  if(ret != 0){  printf("ioctl error\n");  }  for(i=0;i<Test\_MDList.num;i++)  printf("Name:%s,Level:%d,Meid:%d\n",Test\_MDList.MDList[i].MDname,Test\_MDList.MDList[i].MDLevel,Test\_MDList.MDList[i].meid); |

## DefaultCatchall

**SYNOPSIS**

**int cfmSupervisory\_GetCFMDefaultCatchall(uint8 event, uint8 \*value8, uint8 \*value16);**

**int cfmSupervisory\_SetCFMDefaultCatchall(uint8 event, uint8 \*value8);**

**TestCase**

|  |
| --- |
| struct Test\_CFMDefaultCatchall\_st Test\_CFMDefaultCatchall;  memset(&Test\_CFMDefaultCatchall,0,sizeof(struct Test\_CFMDefaultCatchall\_st));  Test\_CFMDefaultCatchall.event=1;  Test\_CFMDefaultCatchall.value8=2;  ret=ioctl(fd,CFM\_SET\_DEFAULTCATCHALL,&Test\_CFMDefaultCatchall);  if(ret != 0){  printf("ioctl error\n");  }  memset(&Test\_CFMDefaultCatchall,0,sizeof(struct Test\_CFMDefaultCatchall\_st));  Test\_CFMDefaultCatchall.event=1;  ret=ioctl(fd,CFM\_GET\_DEFAULTCATCHALL,&Test\_CFMDefaultCatchall);  if(ret != 0){  printf("ioctl error\n");  }  else printf("Test\_CFMDefaultCatchall.value8=%d\n",Test\_CFMDefaultCatchall.value8); |

## DefaultTable

**SYNOPSIS**

**int cfmSupervisory\_GetCFMDefaultTable(void \*param);**

**int cfmSupervisory\_SetCFMDefaultTable(uint16 \*vidList, uint8 event, uint8 value);**

**TestCase**

|  |
| --- |
| struct Test\_CFMDefaultTable\_st Test\_CFMDefaultTable;  struct Test\_DefaultMDLevelEntry\_st DefaultMDLevelEntry;  memset(&DefaultMDLevelEntry,0,sizeof(struct Test\_DefaultMDLevelEntry\_st));  memset(&Test\_CFMDefaultTable,0,sizeof(struct Test\_CFMDefaultTable\_st));  uint16 vidlist[5]={20,34,13,9,22};  Test\_CFMDefaultTable.vidList=&vidlist;  Test\_CFMDefaultTable.event=1;  Test\_CFMDefaultTable.value=3;  ret=ioctl(fd,CFM\_SET\_DEFAULTTABLE,&Test\_CFMDefaultTable);  if(ret != 0){  printf("ioctl error\n");  }  DefaultMDLevelEntry.PrimaryVlanId=12;  ret=ioctl(fd,CFM\_GET\_DEFAULTTALBE,&DefaultMDLevelEntry);  if(ret != 0){  printf("ioctl error\n");  }  else printf("level:%d\n",DefaultMDLevelEntry.Level); |

## Get ChassisManagement

**SYNOPSIS**

**int cfmConfig\_GetChassisManagement(void \* result);**

**TestCase**

|  |
| --- |
| struct Test\_ChassisManagement\_st Test\_ChassisManagement;  memset(&Test\_ChassisManagement,0,sizeof(struct Test\_ChassisManagement\_st));  ret=ioctl(fd,CFM\_GET\_CHASSISMANAGEMENT,&Test\_ChassisManagement);  if(ret != 0){  printf("ioctl error\n");  }  else  {  printf("chassis\_ID:%s\n chassis\_ID\_Length:%u\n chassis\_ID\_Subtype:%u\n management\_Address:%s\n management\_Address\_Domain:%s\n management\_Address\_Domain\_Length:%u\n management\_Address\_Length:%u\n",Test\_ChassisManagement.chassis\_ID,Test\_ChassisManagement.chassis\_ID\_Length,Test\_ChassisManagement.chassis\_ID\_Subtype,Test\_ChassisManagement.management\_Address,Test\_ChassisManagement.management\_Address\_Domain,Test\_ChassisManagement.management\_Address\_Domain\_Length,Test\_ChassisManagement.management\_Address\_Length);  } |

## Peer MEP IDs

**SYNOPSIS**

**int cfmConfig\_SetPeerMEPID(uint16\* peerid, int meid);**

**int cfmConfig\_GetPeerMEPID(uint16\* peerid, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_PeerMEPID\_st Test\_PeerMEPID;  struct Test\_PeerMEPID\_st rTest\_PeerMEPID;  memset(&Test\_PeerMEPID, 0, sizeof(struct Test\_PeerMEPID\_st));  memset(&rTest\_PeerMEPID, 0, sizeof(struct Test\_PeerMEPID\_st));  //set  Test\_PeerMEPID.meid = 1;  for(i=0;i<12;i++){  Test\_PeerMEPID.peerId[i] = i;  }  ret = ioctl(fd, CFM\_SETPEERMEPID, &Test\_PeerMEPID);  if(ret != 0){  printf("ioctl error\n");  }  //get  rTest\_PeerMEPID.meid = 1;  ret = ioctl(fd, CFM\_GETPEERMEPID, &rTest\_PeerMEPID);  if(ret != 0){  printf("ioctl error\n");  }  printf("peerID:\n");  for(i=0;i<12;i++){  printf("%d ", rTest\_PeerMEPID.peerId[i]);  }  printf("\n"); |

# TLV Management

## SenderId TLV Management

### Sender ID Permission

**SYNOPSIS**

**int cfmConfig\_SetSenderIdPermission(int permission, int meid);**

**int cfmConfig\_GetSenderIdPermission(int\* permission, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_SenderIdPermission\_st Test\_SenderIdPermission;  struct Test\_SenderIdPermission\_st rTest\_SenderIdPermission;  memset(&Test\_SenderIdPermission, 0, sizeof(struct Test\_SenderIdPermission\_st));  memset(&rTest\_SenderIdPermission, 0, sizeof(struct Test\_SenderIdPermission\_st));  //set  Test\_SenderIdPermission.meid = 1;  Test\_SenderIdPermission.permission = 2;  ret = ioctl(fd, CFM\_SET\_SENDERIDPERMISSION, &Test\_SenderIdPermission);  if(ret != 0){  printf("ioctl error\n");  }  //get  rTest\_SenderIdPermission.meid = 1;  ret = ioctl(fd, CFM\_GET\_SENDERIDPERMISSION, &rTest\_SenderIdPermission);  if(ret != 0){  printf("ioctl error\n");  }  printf("SenderIdPermission:%d\n", rTest\_SenderIdPermission.permission); |

### Chassis ID Length

**SYNOPSIS**

**int cfmConfig\_SetChassisIDLength(int len);**

**int cfmConfig\_GetChassisIDLength(int\* len);**

### Chassis ID Subtype

**SYNOPSIS**

**int cfmConfig\_SetChassisIDSubtype(int subtype);**

**int cfmConfig\_GetChassisIDSubtype(int\* subtype);**

### Chassis ID

**SYNOPSIS**

**int cfmConfig\_SetChassisID(uint8\* chassisId);**

**int cfmConfig\_GetChassisID(uint8\* chassisId);**

**TestCase**

|  |
| --- |
| int len, subtype, rlen, rsubtype;  len = 10;  subtype = 3;  uint8 ChassisID[50];  uint8 rChassisID[50];  //set  ret = ioctl(fd, CFM\_SET\_CHASSISIDLENGTH, len);  if(ret != 0){  printf("ioctl error\n");  }  ret = ioctl(fd, CFM\_SET\_CHASSISIDSBUTYPE, subtype);  if(ret != 0){  printf("ioctl error\n");  }  for(i=0;i<len;i++){  ChassisID[i] = 0x88;  }  ret = ioctl(fd, CFM\_SET\_CHASSISID, ChassisID);  if(ret != 0){  printf("ioctl error\n");  }  //get  ret = ioctl(fd, CFM\_GET\_CHASSISIDLENGTH, &rlen);  if(ret != 0){  printf("ioctl error\n");  }  ret = ioctl(fd, CFM\_GET\_CHASSISIDSBUTYPE, &rsubtype);  if(ret != 0){  printf("ioctl error\n");  }  printf("rlen:%d, rsubtype:%d\n", rlen, rsubtype);  ret = ioctl(fd, CFM\_GET\_CHASSISID, rChassisID);  if(ret != 0){  printf("ioctl error\n");  }  printf("rChassisID:\n");  for(i=0;i<rlen;i++){  printf("%.2x ", rChassisID[i]);  }  printf("\n"); |

### Management Address Domain Length

**SYNOPSIS**

**int cfmConfig\_SetMgmtAddrDomainLength(int len);**

**int cfmConfig\_GetMgmtAddrDomainLength(int\* len);**

### Management Address Domain

**SYNOPSIS**

**int cfmConfig\_SetMgmtAddrDomain(uint8\* domain);**

**int cfmConfig\_GetMgmtAddrDomain(uint8\* domain);**

**TestCase**

|  |
| --- |
| int len, rlen;  len = 10;  uint8 domain[50];  uint8 rdomain[50];  //set  ret = ioctl(fd, CFM\_SET\_MGMTADDRDOMAINLENGTH, len);  if(ret != 0){  printf("ioctl error\n");  }  for(i=0;i<len;i++){  domain[i] = 0x77;  }  ret = ioctl(fd, CFM\_SET\_MGMTADDRDOMAINM, domain);  if(ret != 0){  printf("ioctl error\n");  }  //get  ret = ioctl(fd, CFM\_GET\_MGMTADDRDOMAINLENGTH, &rlen);  if(ret != 0){  printf("ioctl error\n");  }  printf("rlen:%d\n", rlen);  ret = ioctl(fd, CFM\_GET\_MGMTADDRDOMAINM, rdomain);  if(ret != 0){  printf("ioctl error\n");  }  printf("rdomain:\n");  for(i=0;i<rlen;i++){  printf("%.2x ", rdomain[i]);  }  printf("\n"); |

### Management Address Length

**SYNOPSIS**

**int cfmConfig\_SetMgmtAddrLength(int len);**

**int cfmConfig\_GetMgmtAddrLength(int\* len);**

### Management Address

**SYNOPSIS**

**int cfmConfig\_SetMgmtAddr (uint8\* addr);**

**int cfmConfig\_GetMgmtAddr (uint8\* addr);**

**TestCase**

|  |
| --- |
| int len, rlen;  len = 10;  uint8 addr[50];  uint8 raddr[50];  //set  ret = ioctl(fd, CFM\_SET\_MGMTADDRLENGTH, len);  if(ret != 0){  printf("ioctl error\n");  }  for(i=0;i<len;i++){  addr[i] = 0x66;  }  ret = ioctl(fd, CFM\_SET\_MGMTADDR, addr);  if(ret != 0){  printf("ioctl error\n");  }  //get  ret = ioctl(fd, CFM\_GET\_MGMTADDRLENGTH, &rlen);  if(ret != 0){  printf("ioctl error\n");  }  printf("rlen:%d\n", rlen);  ret = ioctl(fd, CFM\_GET\_MGMTADDR, raddr);  if(ret != 0){  printf("ioctl error\n");  }  printf("raddr:\n");  for(i=0;i<rlen;i++){  printf("%.2x ", raddr[i]);  }  printf("\n"); |

## Organization-Specific TLV Management

### Length

**SYNOPSIS**

**int cfmConfig\_SetOrgSpecificLength(int len);**

**int cfmConfig\_GetOrgSpecificLength(int\* len);**

### OUI

**SYNOPSIS**

**int cfmConfig\_SetOrgSpecificOUI(uint8 \*oui);**

**int cfmConfig\_GetOrgSpecificOUI(uint8\* oui);**

### Sub-Type

**SYNOPSIS**

**int cfmConfig\_SetOrgSpecificSubtype(uint8 subtype);**

**int cfmConfig\_GetOrgSpecificSubtype (uint8\* subtype);**

### Value

**SYNOPSIS**

**int cfmConfig\_SetOrgSpecificValue(uint8\* value);**

**int cfmConfig\_GetOrgSpecificValue(uint8\* value);**

**TestCase**

|  |
| --- |
| uint16 len, rlen;  uint8 oui[3];  uint8 roui[3];  uint8 subtype, rsubtype;  uint8 value[128], rvalue[128];  memset(oui, 0, sizeof(oui));  memset(roui, 0, sizeof(roui));  memset(value, 0, sizeof(value));  memset(rvalue, 0, sizeof(rvalue));  printf("/\*\*\*\*Organization-Specific TLV Management API Test\*\*\*\*/\n");  //set  len = 8;  for(i=0;i<3;i++){  oui[i] = 0x22;  }  subtype = 0x01;  for(i=0;i<len-4;i++){  value[i] = 0x55;  }  ret = ioctl(fd, CFM\_SET\_ORGSPELEN, &len);  if(ret != 0){  printf("ioctl error\n");  }  ret = ioctl(fd, CFM\_SET\_ORGSPEOUI, oui);  if(ret != 0){  printf("ioctl error\n");  }  ret = ioctl(fd, CFM\_SET\_ORGSPESUBTYPE, &subtype);  if(ret != 0){  printf("ioctl error\n");  }  ret = ioctl(fd, CFM\_SET\_ORGSPEVALUE, value);  if(ret != 0){  printf("ioctl error\n");  }  //get  ret = ioctl(fd, CFM\_GET\_ORGSPELEN, &rlen);  if(ret != 0){  printf("ioctl error\n");  }  ret = ioctl(fd, CFM\_GET\_ORGSPEOUI, roui);  if(ret != 0){  printf("ioctl error\n");  }  ret = ioctl(fd, CFM\_GET\_ORGSPESUBTYPE, &rsubtype);  if(ret != 0){  printf("ioctl error\n");  }  printf("os\_len:%.2x, os\_subtype:%.2x\n", rlen, rsubtype);  printf("os\_OUI:\n");  print\_hex\_data(roui, 3);  printf("\n");  ret = ioctl(fd, CFM\_GET\_ORGSPEVALUE, rvalue);  if(ret != 0){  printf("ioctl error\n");  }  printf("os\_value:\n");  print\_hex\_data(rvalue, rlen-4);  printf("\n"); |

# Fault alarm

## Fault alarm threshold

**SYNOPSIS**

**int cfmConfig\_SetAlarmThreshold (int threshold, int meid);**

**int cfmConfig\_GetAlarmThreshold (int\* threshold, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_AlarmPri\_st Test\_AlarmPri;  struct Test\_AlarmPri\_st rTest\_AlarmPri;  memset(&Test\_AlarmPri, 0, sizeof(struct Test\_AlarmPri\_st));  memset(&rTest\_AlarmPri, 0, sizeof(struct Test\_AlarmPri\_st));  //set  Test\_AlarmPri.meid = 1;  Test\_AlarmPri.AlarmPri = 2;  ret = ioctl(fd, CFM\_SET\_ALARMTHRESHOLD, &Test\_AlarmPri);  if(ret != 0){  printf("ioctl error\n");  }  //get  rTest\_AlarmPri.meid = 1;  ret = ioctl(fd, CFM\_GET\_ALARMTHRESHOLD, &rTest\_AlarmPri);  if(ret != 0){  printf("ioctl error\n");  }  printf("AlarmPri:%d\n", rTest\_AlarmPri.AlarmPri); |

# CCP Management

## Continuity Check module enable

**SYNOPSIS**

**int cfmConfig\_SetCCIenabled(int CCIenabled, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_CCIenabled\_st Test\_CCIenabled;  memset(&Test\_CCIenabled, 0, sizeof(struct Test\_CCIenabled\_st));  Test\_CCIenabled.meid = 1;  Test\_CCIenabled.CCIenabled = 1;  ret = ioctl(fd, CFM\_SET\_CCIENABLED, &Test\_CCIenabled);  if(ret != 0){  printf("ioctl error\n");  }  sleep(5);  Test\_CCIenabled.CCIenabled = 0;  ret = ioctl(fd, CFM\_SET\_CCIENABLED, &Test\_CCIenabled);  if(ret != 0){  printf("ioctl error\n");  } |

## CCM interval

**SYNOPSIS**

**int cfmConfig\_SetCCMInterval(int interval, int meid);**

**int cfmConfig\_GetCCMInterval(int interval, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_CCMInterval\_st Test\_CCMInterval;  struct Test\_CCMInterval\_st rTest\_CCMInterval;  memset(&Test\_CCMInterval, 0, sizeof(struct Test\_CCMInterval\_st));  memset(&rTest\_CCMInterval, 0, sizeof(struct Test\_CCMInterval\_st));  //set CCMInterval  Test\_CCMInterval.meid = 1;  Test\_CCMInterval.interval = 4;  ret = ioctl(fd, CFM\_SET\_CCMINTERVAL, &Test\_CCMInterval);  if(ret != 0){  printf("ioctl error\n");  }  //get CCMInterval  rTest\_CCMInterval.meid = 1;  ret = ioctl(fd, CFM\_GET\_CCMINTERVAL, &rTest\_CCMInterval);  if(ret != 0){  printf("ioctl error\n");  }  printf("CCMInterval:%d\n", rTest\_CCMInterval.interval); |

## Received CCM PDU

**SYNOPSIS**

**int cfmConfig\_GetLastRcvdErrorCCM (uint8\* ccm, int \*len, int meid);**

**int cfmConfig\_GetLastRcvdXconCCM (uint8\* ccm, int \*len, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_ccm\_st Test\_ccm;  //get last recv error ccm  memset(&Test\_ccm, 0, sizeof(struct Test\_ccm\_st));  Test\_ccm.meid = 1;  ret = ioctl(fd, CFM\_GET\_LASTRCVDERRORCCM, &Test\_ccm);  if(ret != 0){  printf("ioctl error\n");  }  printf("last recv error ccm:\n");  print\_hex\_data(Test\_ccm.ccm, Test\_ccm.len);  //get last recv xcon ccm  memset(&Test\_ccm, 0, sizeof(struct Test\_ccm\_st));  Test\_ccm.meid = 1;  ret = ioctl(fd, CFM\_GET\_LASTRCVDXCONCCM, &Test\_ccm);  if(ret != 0){  printf("ioctl error\n");  }  printf("last recv xcon ccm:\n");  print\_hex\_data(Test\_ccm.ccm, Test\_ccm.len); |

## Continuity Check counters

**SYNOPSIS**

**int cfmConfig\_GetOutOfSeqCCMCounts (int\* counts, int meid);**

**int cfmConfig\_GetXmitCCMCounts (int\* counts, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_CCCount\_st Test\_CCCount;  //get OutOfSeqCCMCounts  memset(&Test\_CCCount, 0, sizeof(struct Test\_CCCount\_st));  Test\_CCCount.meid = 1;  ret = ioctl(fd, CFM\_GET\_OUTOFSEQCCMCOUNTS, &Test\_CCCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("OutOfSeqCCMCounts:%d\n", Test\_CCCount.count);  //get XmitCCMCounts  memset(&Test\_CCCount, 0, sizeof(struct Test\_CCCount\_st));  Test\_CCCount.meid = 1;  ret = ioctl(fd, CFM\_GET\_XMITCCMCOUNTS, &Test\_CCCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("XmitCCMCounts:%d\n", Test\_CCCount.count); |

## CCMdatabase node creation

**SYNOPSIS**

**int cfmConfig\_CreateCCMDatabaseNode(uint16 MEPid, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_CreateCCMDatabaseNode\_st Test\_CreateCCMDatabaseNode;  memset(&Test\_CreateCCMDatabaseNode, 0, sizeof(struct Test\_CreateCCMDatabaseNode\_st));  Test\_CreateCCMDatabaseNode.meid = 1;  Test\_CreateCCMDatabaseNode.MEPid = 2;  ret = ioctl(fd, CFM\_CREATECCMDABASENODE, &Test\_CreateCCMDatabaseNode);  if(ret != 0){  printf("ioctl error\n");  } |

## TLV management

### Port Status TLV management

**SYNOPSIS**

**int cfmConfig\_SetPortStatus(uint8 value, int meid);**

**int cfmConfig\_GetPortStatus(uint8\* value,int meid);**

### Interface Status TLV management

**SYNOPSIS**

**int cfmConfig\_SetInterfaceStatus(uint8 value, int meid);**

**int cfmConfig\_GetInterfaceStatus(uint8\* value,int meid);**

**TestCase**

|  |
| --- |
| struct Test\_Status\_st Test\_Status;  struct Test\_Status\_st rTest\_Status;  //PortStatus  memset(&Test\_Status, 0, sizeof(struct Test\_Status\_st));  memset(&rTest\_Status, 0, sizeof(struct Test\_Status\_st));  //set  Test\_Status.meid = 1;  Test\_Status.value = 0x02;  ret = ioctl(fd, CFM\_SET\_PORTSTATUS, &Test\_Status);  if(ret != 0){  printf("ioctl error\n");  }  //get  rTest\_Status.meid = 1;  ret = ioctl(fd, CFM\_GET\_PORTSTATUS, &rTest\_Status);  if(ret != 0){  printf("ioctl error\n");  }  printf("PortStatus:%.2x\n", rTest\_Status.value);  //InterfaceStatus  memset(&Test\_Status, 0, sizeof(struct Test\_Status\_st));  memset(&rTest\_Status, 0, sizeof(struct Test\_Status\_st));  //set  Test\_Status.meid = 1;  Test\_Status.value = 0x02;  ret = ioctl(fd, CFM\_SET\_INTERFASESTATUS, &Test\_Status);  if(ret != 0){  printf("ioctl error\n");  }  //get  rTest\_Status.meid = 1;  ret = ioctl(fd, CFM\_GET\_INTERFASESTATUS, &rTest\_Status);  if(ret != 0){  printf("ioctl error\n");  }  printf("InterfaceStatus:%.2x\n", rTest\_Status.value); |

### Organization-Specific TLV

**SYNOPSIS**

**int cfmConfig\_SetCCMOrgSpecificPermission(int permission, int meid);**

**int cfmConfig\_GetCCMOrgSpecificPermission(int\* permission, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_OrgSpePermission\_st Test\_OrgSpePermission;  struct Test\_OrgSpePermission\_st rTest\_OrgSpePermission;  memset(&Test\_OrgSpePermission, 0, sizeof(struct Test\_OrgSpePermission\_st));  memset(&rTest\_OrgSpePermission, 0, sizeof(struct Test\_OrgSpePermission\_st));  //set  Test\_OrgSpePermission.meid = 1;  Test\_OrgSpePermission.permission = 1;  ioctl(fd, CFM\_SET\_CCORGSPEPERMISSION, &Test\_OrgSpePermission);  //get  rTest\_OrgSpePermission.meid = 1;  ioctl(fd, CFM\_GET\_CCORGSPEPERMISSION, &rTest\_OrgSpePermission);  printf("CC OrgSpe Permission:%d\n", rTest\_OrgSpePermission.permission); |

# LTP Management

## Linktrace module Management

**SYNOPSIS**

**int cfmConfig\_LinkTrace\_Start(uint8\* targetAddr,int meid)**

**TestCase**

|  |
| --- |
| struct Test\_LTStart\_st Test\_LTStart;  memset(&Test\_LTStart, 0, sizeof(struct Test\_LTStart\_st));  Test\_LTStart.meid = 1;  memcpy(Test\_LTStart.MacAddr, hostmac\_eth0, ADDR\_LEN);  ret = ioctl(fd, CFM\_LT\_START, &Test\_LTStart);  if(ret != 0){  printf("ioctl error\n");  } |

**SYNOPSIS**

**int cfmConfig\_LinkTrace\_GetResult(void\* list, uint8\* targetAddr, int meid)**

**TestCase**

|  |
| --- |
| struct result\_param result;  int i=0,j=0;  char Dest\_mac[6] = {0x00,0x0C,0x29,0xF1,0x0C,0xC6};  memset(&result,0, sizeof(struct result\_param));  result.meid = 1;  memcpy(result.mac,Dest\_mac,ADDR\_LEN);  ret = ioctl(fd,CFM\_LT\_GETRESULT, &result);  if(ret != 0)  {  printf("ioctl error!\n");  }  if(ret == 1)  printf("link trace's node number is beyond 10, so not all node printed!\n");  printf("LT result:\n");  printf("mac\_addr TTL Ingress Action IngressMac Egress Action EgressMac \n");  for(;i< result.node\_num;i++)  {  for(;j<ADDR\_LEN;j++)  printf("%2.2x",result.node[i].mac\_addr);  printf(" %d",result.node[i].TTL);  printf(" %d", result.node[i].ingress\_action);  printf(" ");  for(j=0;j<ADDR\_LEN;j++)  printf("%2.2x", result.node[i].ingressMAC);    printf(" %d",result.node[i].egress\_action);  for(j=0;j<ADDR\_LEN;j++)  printf("%2.2x", result.node[i].egressMAC);    printf("\n");    } |

## Linktrace Counters

**SYNOPSIS**

**int cfmConfig\_GetXmitLTRCounts (int\* counts, int meid)**

**int cfmConfig\_GetXmitLTMCounts(int\* counts, int meid)**

**int cfmConfig\_GetLTMreceivedCounts(int\* counts, int meid)**

**int cfmConfig\_GetLTRreceivedCounts(int\* counts, int meid)**

**int cfmConfig\_GetUnexpectedLTRCounts (int\* counts, int meid)**

**TestCase**

|  |
| --- |
| struct Test\_LTCount\_st Test\_LTCount;  //xmit LTR  memset(&Test\_LTCount, 0, sizeof(struct Test\_LTCount\_st));  Test\_LTCount.meid = 1;  ret = ioctl(fd, CFM\_LT\_GETXMITLTRCOUNTS, &Test\_LTCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("xmit LTR:%d\n", Test\_LTCount.count);  //xmit LTM  memset(&Test\_LTCount, 0, sizeof(struct Test\_LTCount\_st));  Test\_LTCount.meid = 1;  ret = ioctl(fd, CFM\_LT\_GETXMITLTMCOUNTS, &Test\_LTCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("xmit LTM:%d\n", Test\_LTCount.count);  //recv LTM  memset(&Test\_LTCount, 0, sizeof(struct Test\_LTCount\_st));  Test\_LTCount.meid = 1;  ret = ioctl(fd, CFM\_LT\_GETLTMRECEIVEDCOUNTS, &Test\_LTCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("recv LTM:%d\n", Test\_LTCount.count);  //recv LTR  memset(&Test\_LTCount, 0, sizeof(struct Test\_LTCount\_st));  Test\_LTCount.meid = 1;  ret = ioctl(fd, CFM\_LT\_GETLTRRECEIVEDCOUNTS, &Test\_LTCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("recv LTR:%d\n", Test\_LTCount.count);  //unexpect LTR  memset(&Test\_LTCount, 0, sizeof(struct Test\_LTCount\_st));  Test\_LTCount.meid = 1;  ret = ioctl(fd, CFM\_LT\_GETUNEXPECTEDLTRCOUNTS, &Test\_LTCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("unexpect LTR:%d\n", Test\_LTCount.count); |

## LT PDU Set

**SYNOPSIS**

**int cfmConfig\_SetNextLTTranID (int\* ltid, int meid)**

**int cfmConfig\_GetNextLTTranID (int\* ltid, int meid)**

**TestCase**

|  |
| --- |
| //get  struct Test\_LTTransId\_st Test\_LTTransId;  memset(&Test\_LTTransId, 0, sizeof(struct Test\_LTTransId\_st));  Test\_LTTransId.meid = 1;  ret = ioctl(fd, CFM\_LT\_GETNEXTTRANSID, &Test\_LTTransId);  if(ret != 0){  printf("ioctl error\n");  }  printf("Next LTTransId:%d\n", Test\_LTTransId.ltid);  //set  memset(&Test\_LTTransId, 0, sizeof(struct Test\_LTTransId\_st));  Test\_LTTransId.meid = 1;  Test\_LTTransId.ltid=10;  ret = ioctl(fd, CFM\_LT\_SETNEXTTRANSID, &Test\_LTTransId);  if(ret != 0){  printf("ioctl error\n");  } |

**SYNOPSIS**

**int cfmConfig\_SetLTMTTL(uint8 ttl, int meid)**

**TestCase**

|  |
| --- |
| struct Test\_LTSetLTMTtl\_st Test\_LTSetLTMTtl;  memset(&Test\_LTSetLTMTtl, 0, sizeof(struct Test\_LTSetLTMTtl\_st));  Test\_LTSetLTMTtl.meid = 1;  Test\_LTSetLTMTtl.ttl = 64;  ret = ioctl(fd, CFM\_LT\_SETLTMTTL, &Test\_LTSetLTMTtl);  if(ret != 0){  printf("ioctl error\n");  } |

**SYNOPSIS**

**int cfmConfig\_SetLTMflags(uint8 flags, int meid)**

**TestCase**

|  |
| --- |
| struct Test\_LTSetLTMFlags\_st Test\_LTSetLTMFlags;  memset(&Test\_LTSetLTMFlags, 0, sizeof(struct Test\_LTSetLTMFlags\_st));  Test\_LTSetLTMFlags.meid = 1;  Test\_LTSetLTMFlags.flags = 0x80; //useFDBonly  ret = ioctl(fd, CFM\_LT\_SETLTMFLAGS, &Test\_LTSetLTMFlags);  if(ret != 0){  printf("ioctl error\n");  } |

**SYNOPSIS**

**int cfmConfig\_SetLTMSenderIDPermission(int flags, int meid);**

**int cfmConfig\_GetLTMSenderIDPermission(int\* flags, int meid)**

**TestCase**

|  |
| --- |
| struct Test\_LTSenderIDPermission\_st permission;  permission.flags = 1;  permission.meid = 1;  //set  ret = ioctl(fd,CFM\_SETLTM\_SENDERID\_PERMISSION, &permission);  if(ret != 0)  printf("ioctl error\n");  //get  permission.meid = 1;  ret = ioctl(fd,CFM\_GETLTM\_SENDERID\_PERMISSION, &permission);  if(ret != 0)  printf("ioctl error\n");  else  printf("LTM SenderID permission :%d\n", permission.flags); |

**SYNOPSIS**

**int cfmConfig\_SetLTRSenderIDPermission(int flags, int meid);**

**int cfmConfig\_GetLTRSenderIDPermission(int\* flags, int meid)**

**TestCase**

|  |
| --- |
| struct Test\_LTSenderIDPermission\_st permission;  memset(&permission, 0, sizeof(struct Test\_LTSenderIDPermission\_st));  permission.flags = 1;  permission.meid = 1;  //set  ret = ioctl(fd,CFM\_SETLTR\_SENDERID\_PERMISSION, &permission);  if(ret != 0)  printf("ioctl error\n");  //get  memset(&permission, 0, sizeof(struct Test\_LTSenderIDPermission\_st));  permission.meid = 1;  ret = ioctl(fd,CFM\_GETLTR\_SENDERID\_PERMISSION, &permission);  if(ret != 0)  printf("ioctl error\n");  else  printf("LTR SenderID permission :%d\n", permission.flags);  } |

## Egress ID

**SYNOPSIS**

**int cfmConfig\_SetEgressID(uint8\* gressid, int meid)**

**int cfmConfig\_GetEgressID(uint8\* gressid, int meid)**

**TestCase**

|  |
| --- |
| struct Test\_LTEgressId\_st Test\_LTEgressId;  struct Test\_LTEgressId\_st rTest\_LTEgressId;  memset(&Test\_LTEgressId, 0, sizeof(struct Test\_LTEgressId\_st));  memset(&rTest\_LTEgressId, 0, sizeof(struct Test\_LTEgressId\_st));  //set  Test\_LTEgressId.meid = 1;  Test\_LTEgressId.gressid[0] = 0x00;  Test\_LTEgressId.gressid[1] = 0x11;  memcpy(Test\_LTEgressId.gressid+2, hostmac\_eth0, ADDR\_LEN);  ret = ioctl(fd, CFM\_SETEGRESSID, &Test\_LTEgressId);  if(ret != 0){  printf("ioctl error\n");  }  //get  rTest\_LTEgressId.meid = 1;  ret = ioctl(fd, CFM\_GETEGRESSID, &rTest\_LTEgressId);  if(ret != 0){  printf("ioctl error\n");  }  printf("EgressId:\n");  for(i=0;i<8;i++){  printf("%.2x ", rTest\_LTEgressId.gressid[i]);  }  printf("\n"); |

# LBP Management

## Loopback Module

**SYNOPSIS**

**int cfmConfig\_Loopback\_Start(int lbmstosend,uint8 \*desAddr,int meid);**

**TestCase**

|  |
| --- |
| struct Test\_LBStart\_st Test\_LBStart;  uint8 tarAddr[ADDR\_LEN]={0x00,0x0C,0c29,0x9f,0x98,0xd7};  memset(&Test\_LBStart, 0, sizeof(struct Test\_LBStart\_st));  Test\_LBStart.meid = 1;  //to do  memcpy(Test\_LBStart.MacAddr,tarAddr , ADDR\_LEN);  Test\_LBStart.lbmstosend = 5;  ret = ioctl(fd, CFM\_LB\_START, &Test\_LBStart);  if(ret != 0){  printf("ioctl error\n");  } |

**SYNOPSIS**

**int cfmConfig\_LoopBack\_GetResult(int\*result,int meid);**

**TestCase**

|  |
| --- |
| struct Test\_LBGetResult\_st Test\_LBGetResult;  memset(&Test\_LBGetResult, 0, sizeof(struct Test\_LBGetResult\_st));  Test\_LBGetResult.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETRESULT, &Test\_LBGetResult);  if(ret != 0){  printf("ioctl error\n");  }  printf("LB result:%d\n", Test\_LBGetResult.result); |

**SYNOPSIS**

**int cfmConfig\_SetLBAvaliable(int valid,int meid);**

**int cfmConfig\_GetLBAvaliable(int\*valid,int meid);**

**TestCase**

|  |
| --- |
| struct Test\_LBAvaliable\_st Test\_LBAvaliable;  struct Test\_LBAvaliable\_st rTest\_LBAvaliable;  memset(&Test\_LBAvaliable, 0, sizeof(struct Test\_LBAvaliable\_st));  memset(&rTest\_LBAvaliable, 0, sizeof(struct Test\_LBAvaliable\_st));  //set LBAvaliable  Test\_LBAvaliable.meid = 1;  Test\_LBAvaliable.LBAvaliable = 1;  ret = ioctl(fd, CFM\_LB\_SETLBAVALIABLE, &Test\_LBAvaliable);  if(ret != 0){  printf("ioctl error\n");  }  //get LBAvaliable  rTest\_LBAvaliable.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETLBAVALIABLE, &rTest\_LBAvaliable);  if(ret != 0){  printf("ioctl error\n");  }  printf("LBAvaliable:%d\n", rTest\_LBAvaliable.LBAvaliable); |

## Loopback Counters

**SYNOPSIS**

**int cfmConfig\_GetErrorRcvdLBRCounts(int\*count, int meid);**

**int cfmConfig\_GetCorrectRcvdLBRCounts(int\*count, int meid);**

**int cfmConfig\_GetXmitLBMCounts(int\*count, int meid);**

**int cfmConfig\_GetXmitLBRCounts(int\*count, int meid);**

**int cfmConfig\_GetRcvdLBMCounts(int\*count, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_LBCount\_st Test\_LBCount;  //ErrorRcvdLBRCounts  memset(&Test\_LBCount, 0, sizeof(struct Test\_LBCount\_st));  Test\_LBCount.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETERRORRCVDLBRCOUNTS, &Test\_LBCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("ErrorRcvdLBRCounts:%d\n", Test\_LBCount.count);  //CorrectRcvdLBRCounts  memset(&Test\_LBCount, 0, sizeof(struct Test\_LBCount\_st));  Test\_LBCount.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETCORRECTRCVDLBRCOUNTS, &Test\_LBCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("CorrectRcvdLBRCounts:%d\n", Test\_LBCount.count);  //XmitLBMCounts  memset(&Test\_LBCount, 0, sizeof(struct Test\_LBCount\_st));  Test\_LBCount.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETXMITLBMCOUNTS, &Test\_LBCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("XmitLBMCounts:%d\n", Test\_LBCount.count);  //XmitLBRCounts  memset(&Test\_LBCount, 0, sizeof(struct Test\_LBCount\_st));  Test\_LBCount.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETXMITLBRCOUNTS, &Test\_LBCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("XmitLBRCounts:%d\n", Test\_LBCount.count);  //RcvdLBMCounts  memset(&Test\_LBCount, 0, sizeof(struct Test\_LBCount\_st));  Test\_LBCount.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETRCVDLBMCOUNTS, &Test\_LBCount);  if(ret != 0){  printf("ioctl error\n");  }  printf("RcvdLBMCounts:%d\n", Test\_LBCount.count); |

## Transaction ID

**SYNOPSIS**

**int cfmConfig\_GetNextLBTransID(int\* lbid, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_LBTransID\_st Test\_LBTransID;  memset(&Test\_LBTransID, 0, sizeof(struct Test\_LBTransID\_st));  Test\_LBTransID.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETNEXTLBTRANSID, &Test\_LBTransID);  if(ret != 0){  printf("ioctl error\n");  }  printf("NextLBTransID:%d\n", Test\_LBTransID.lbid); |

## PDU Stored

**SYNOPSIS**

**int cfmConfig\_GetLastRcvdLBM(uint8\*pkt\_data, uint32\*pkt\_len, int meid);**

**int cfmConfig\_GetLastRcvdLBR(uint8\*pkt\_data, uint32\*pkt\_len, int meid);**

**int cfmConfig\_GetLastSentLBM(uint8\*pkt\_data, uint32\*pkt\_len, int meid);**

**int cfmConfig\_GetLastSentLBR(uint8\*pkt\_data, uint32\*pkt\_len, int meid);**

**TestCase**

|  |
| --- |
| struct Test\_LBPdu\_st Test\_LBPdu;  //LastRcvdLBM  memset(&Test\_LBPdu, 0, sizeof(struct Test\_LBPdu\_st));  Test\_LBPdu.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETLASTRCVDLBM, &Test\_LBPdu);  if(ret != 0){  printf("ioctl error\n");  }  printf("LastRcvdLBM:\n");  print\_hex\_data(Test\_LBPdu.pkt\_data, Test\_LBPdu.pkt\_len);  //LastRcvdLBR  memset(&Test\_LBPdu, 0, sizeof(struct Test\_LBPdu\_st));  Test\_LBPdu.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETLASTRCVDLBR, &Test\_LBPdu);  if(ret != 0){  printf("ioctl error\n");  }  printf("LastRcvdLBR:\n");  print\_hex\_data(Test\_LBPdu.pkt\_data, Test\_LBPdu.pkt\_len);  }  struct Test\_LBPdu\_st Test\_LBPdu;  //LastSentLBM  memset(&Test\_LBPdu, 0, sizeof(struct Test\_LBPdu\_st));  Test\_LBPdu.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETLASTSENTLBM, &Test\_LBPdu);  if(ret != 0){  printf("ioctl error\n");  }  printf("LastSentLBM:\n");  print\_hex\_data(Test\_LBPdu.pkt\_data, Test\_LBPdu.pkt\_len);  //LastSentLBR  memset(&Test\_LBPdu, 0, sizeof(struct Test\_LBPdu\_st));  Test\_LBPdu.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETLASTRSENTLBR, &Test\_LBPdu);  if(ret != 0){  printf("ioctl error\n");  }  printf("LastSentLBR:\n");  print\_hex\_data(Test\_LBPdu.pkt\_data, Test\_LBPdu.pkt\_len);  } |

## Timeout

**SYNOPSIS**

**int cfmConfig\_SetLBTimeOut(uint32 time,int meid);**

**int cfmConfig\_GetLBTimeOut(uint32\*time,int meid);**

**TestCase**

|  |
| --- |
| struct Test\_LBTimeout\_st Test\_LBTimeout;  struct Test\_LBTimeout\_st rTest\_LBTimeout;  memset(&Test\_LBTimeout, 0, sizeof(struct Test\_LBTimeout\_st));  memset(&rTest\_LBTimeout, 0, sizeof(struct Test\_LBTimeout\_st));  //set  Test\_LBTimeout.meid = 1;  Test\_LBTimeout.time = 5;  ret = ioctl(fd, CFM\_LB\_SETTIMEOUT, &Test\_LBTimeout);  if(ret != 0){  printf("ioctl error\n");  }  //get  rTest\_LBTimeout.meid = 1;  ret = ioctl(fd, CFM\_LB\_GETTIMEOUT, &Test\_LBTimeout);  if(ret != 0){  printf("ioctl error\n");  }  printf("LB Timeout:%d\n", rTest\_LBTimeout.time); |