4576 Operations and Programming Overview

Basic sequence of events

1. Query (probe) the driver for configuration status, driver, adapter, link(mtu) and connection information.
2. If the driver has not already been configured:Parse the configuration file
3. Initialize the interface
4. Open connections with desired traffic shaping parameters
5. Transmit and Receive data
6. Query statistics
7. Close connections
8. Exit

Query (probe) for driver, adapter, link and connection information

The purpose of this operation is for the application to determine the operating environment of the driver and adapter. This information will disclose the adapter hardware parameters, the driver version, board count and if the driver has already been configured. If the driver has been configured then the current mtu size and the number of open connections already in service can be determined.
NOTE: Any defines used in the subsequent source code examples can be found in isar_api.h included in the driver package.

Query interface

The query API prototype: **u32 isar_query (void *query_s);**
The query API structure type:
The isar_hdr_s structure is common to all query and control API.

```
typedef struct isar_hdr_s {          /* common header for all query requests */
      U32           status;            /* completion status          */
      U32           type;              /* query type code            */
      U32           link;              /* link number (where applic) */
      U32           bytesOut;          /* api->drv buffer length     */
      U32           bytesIn;           /* drv->api buffer length     */
      ISAR_USR_TAG  tag;               /* opaque user tag/id         */
} isar_hdr_t;
typedef struct isar_query_s {

   isar_hdr_t       hdr;               /* request header             */

   union {

      struct query_drvr_s {
         char  vendor[ISAR_ILEN];   /* out - vendor               */
         char  desc[ISAR_ILEN];     /* out - description xx75-SAR */
         char  environ[ISAR_ILEN];  /* out - environ OS-PROC_BIT  */
         char  version[ISAR_ILEN];  /* out - release/build number */
         U32   adapters;            /* out - adapter count        */
      } driver;

      struct query_adap_s {
         U32   adapter;             /* in  - adapter index        */
         char  vendor[ISAR_ILEN];   /* out - manufactor           */
```

```
   char  model[ISAR_ILEN];    /* out - model string          */
   char  bus[ISAR_ILEN];      /* out - bustype string        */
   char  name[ISAR_ILEN];     /* out - name string           */
   U32   devid;               /* out - device id             */
   U32   subid;               /* out - subsystem id          */
   U32   class;               /* out - class/rev             */
   U32   hwrev;               /* out - class/rev             */
   U32   serial;              /* out - serial number         */
   U32   ports;               /* out - onboard memory size   */
   U32   sram;                /* out - onboard memory size   */
   U32   sar;                 /* out - SAR type/rev          */
   U32   phy;                 /* out - SAR type/rev          */
   U8    umac[ISAR_MAC_LEN];  /* out - user MAC address      */
   U8    fmac[ISAR_MAC_LEN];  /* out - factory MAC address   */
} adapter;

struct query_port_s {
   U32   adapter;             /* in  - adapter index         */
   U32   port;                /* in  - port index            */
   U32   type;                /* out - phy type/rev          */
   U32   speed;               /* out - rate (Mbits/sec)      */
   U32   connector;           /* out - rate (Mbits/sec)      */
} port;

struct query_link_s {
   U32   adapter;             /* out - adapter number        */
   U32   port;                /* out - port number           */
   U32   state;               /* out - link up (true,false)  */
   U32   mtu;                 /* out - max packet size       */
   U32   segVccs;             /* out - maximum total VCs     */
   U32   rsmVccs;             /* out - maximum total VCs     */
   U32   activeVcs;           /* out - active VCs            */
   U32   segQsize;            /* out - maximum total VCs     */
   U32   rsmQsize;            /* out - maximum total VCs     */
} link;

struct query_lstat_s {
   U32    aal0In;             /* out - counter               */
   U32    aal0Out;            /* out - counter               */
   U32    pcktsIn;            /* out - counter               */
   U32    pcktsOut;           /* out - counter               */
   U32    octetsIn;           /* out - counter               */
   U32    octetsOut;          /* out - counter               */
   U32    oamIn;              /* out - counter               */
   U32    oamOut;             /* out - counter               */
   U32    discards;           /* out - counter               */
   U32    losCount;           /* out - counter               */
} lstats;

struct query_vc_s {
   U32         actvc;         /* i/o - active VCC index      */
```

```
        ISAR_VCID     vcid;          /* i/o - VCID handle          */
        U32           slot;           /* out - VCC table slot number  */
        U32           state;          /* out - vc state             */
        U32           vci;            /* out - table index's VCI    */
        U32           vpi;            /* out - table index's VPI    */
        U32           type;           /* out - AAL type             */
        U32           mode;           /* out - mode (UBR, VBR, CBR  */
        U32           tqdepth;        /* out - TxQ threshold        */
        ISAR_RCV_IND *rcv_ind;        /* out - rx callback          */
        ISAR_IND_ARG *rcv_arg;        /* out - rx callback argument */
      } vcinfo;

      struct query_vstat_s {
        ISAR_VCID  vcid;              /* in  - vcid handle          */
        U32        aal0In;            /* out - counter              */
        U32        aal0Out;           /* out - counter              */
        U32        pcktsIn;           /* out - counter              */
        U32        pcktsOut;          /* out - counter              */
        U32        octetsIn;          /* out - counter              */
        U32        octetsOut;         /* out - counter              */
        U32        oamIn;             /* out - counter              */
        U32        oamOut;            /* out - counter              */
        U32        discards;          /* out - counter              */
      } vstats;
    } uq;
} isar_query_t;
```

Driver Information

Obtaining the driver information is the first step in that the user can determine that the revision is correct and especially the number of cards in the system.
A typical print out of the information would look like:

```
Driver Information
vendor : Interphase
desc   : x576 iSAR API driver
env    : SunOS 5.8 sparc
version: X01 build mre.3696
bd cnt : 1
```

and be produced by the following code:

```
static u32
get_driver_info() {
        u32 err;
        isar_query_t info;

        info.hdr.status   = NULL_STATUS;
        info.hdr.link     =  0;
        info.hdr.bytesIn  = sizeof(info);
        info.hdr.bytesOut = sizeof(info);
        info.hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        info.hdr.type     = ISAR_DRVR_INFO;
        err = isar_query(&info);
        if((err == ISAR_PASS) && (info.hdr.status == IA_ERR_NONE)) {
                sprintf(stdout, "Driver Information\n");
                sprintf(stdout, "vendor : %s\n",&info.uq.driver.vendor);
                sprintf(stdout, "desc   : %s\n",&info.uq.driver.desc);
                sprintf(stdout, "env    : %s\n",&info.uq.driver.environ)
                sprintf(stdout, "version: %s\n",&info.uq.driver.version);
                sprintf(stdout, "bd cnt : %d\n",info.uq.driver.adapters);
        }
        return(info.uq.drivers.adapters);
}
```

Adapter Information

A typical output for the adapter information would look like:

```
Adapter Information
bd#    : 0
vendor : Interphase
model  : 4576
bus    : PMC
name   : SX00653-X01          (Interphase part number for the driver package)
devid  : 0x823414f1           (PCI vendor and device ID for the Conexant chipset)
subid  : 0x10107e             (PCI subsystem and subvendor ID, Interphase)
class  : 0x2030005
hwrev  : 0x10
serial : 0x0
ports  : 0x1
sram   : 0x401800             (memory space occupied by card)
sar    : 0x5                  (SAR revision from the chip)
phy    : 0x77                 (PHY revision from the chip)
umac   : 0000000000000000
fmac   : 0000000000000000
```

The output above would be produced from the below sample code:

```
static void
get_adapter_info(u32 bdnum, u32 bdlink) {
        u32 err, x;
        isar_query_t info;

        info.hdr.status   = NULL_STATUS;
        info.hdr.link     = bdlink;
        info.hdr.bytesIn  = sizeof(info);
        info.hdr.bytesOut = sizeof(info);
        info.hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        info.hdr.type     = ISAR_ADAP_INFO;
        info.uq.adapter.adapter = bdnum;
        err = isar_query(&info);
        if((err == ISAR_PASS) && (info.hdr.status == IA_ERR_NONE)) {
                sprintf(stdout, "Adapter Information\n");
                sprintf(stdout, "bd#    : %d\n",info.uq.adapter.adapter);
                sprintf(stdout, "vendor : %s\n",&info.uq.adapter.vendor);
                sprintf(stdout, "model  : %s\n",&info.uq.adapter.model);
                sprintf(stdout, "bus    : %s\n",&info.uq.adapter.bus);
                sprintf(stdout, "name   : %s\n",&info.uq.adapter.name);
                sprintf(stdout, "devid  : 0x%x\n",info.uq.adapter.devid);
                sprintf(stdout, "subid  : 0x%x\n",info.uq.adapter.subid);
                sprintf(stdout, "class  : 0x%x\n",info.uq.adapter.class);
                sprintf(stdout, "hwrev  : 0x%x\n",info.uq.adapter.hwrev);
                sprintf(stdout, "serial : 0x%x\n",info.uq.adapter.serial);
                sprintf(stdout, "ports  : 0x%x\n",info.uq.adapter.ports);
                sprintf(stdout, "sram   : 0x%x\n",info.uq.adapter.sram);
                sprintf(stdout, "sar    : 0x%x\n",info.uq.adapter.sar);
                sprintf(stdout, "phy    : 0x%x\n",info.uq.adapter.phy);
                sprintf(stdout, "umac   : ");
```

```
                for(x = 0; x < ISAR_MAC_LEN; x++) {
                        sprintf(stdout, "%x", info.uq.adapter.umac[x]);
                }
                sprintf(stdout, "\nfmac   : ");
                for(x = 0; x < ISAR_MAC_LEN; x++) {
                        sprintf(stdout, "%x", info.uq.adapter.fmac[x]);
                }
        }
}
```

Link Information

A typical output for the adapter information would look like:

```
Link Information
bd#    : 0
port   : 0
state  : 4095
mtu    : 4096
SegVcc : 32768
RsmVcc : 30976
ActVcc : 0
SegQsz : 1024
RsmQsz : 1024
```

The return information from this query is best saved in a global struct for the application. Values such as the mtu size will be needed when sending data to the driver.

The output above would be produced from the below sample code:

```
static void
get_link_info(u32 adap,  u32 bdlink) {
        u32 err;
        isar_query_t info;

        info.hdr.status   = NULL_STATUS;
        info.hdr.link     = bdlink;
        info.hdr.bytesIn  = sizeof(info);
        info.hdr.bytesOut = sizeof(info);
        info.hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        info.hdr.type     = ISAR_LINK_INFO;
        err = isar_query(&info);
        if((err == ISAR_PASS) && (info.hdr.status == IA_ERR_NONE)) {
                sprintf(stdout, "Link Information\n");
                sprintf(stdout, "bd#    : %d\n",info.uq.link.adapter);
                sprintf(stdout, "port   : %d\n",info.uq.link.port);
                sprintf(stdout, "state  : %d\n",info.uq.link.state);
                sprintf(stdout, "mtu    : %d\n",info.uq.link.mtu);
                sprintf(stdout, "SegVcc : %d\n",info.uq.link.segVccs);
                sprintf(stdout, "RsmVcc : %d\n",info.uq.link.rsmVccs);
                sprintf(stdout, "ActVcc : %d\n",info.uq.link.activeVcs);
                sprintf(stdout, "SegQsz : %d\n",info.uq.link.segQsize);
                sprintf(stdout, "RsmQsz : %d\n",info.uq.link.rsmQsize);
                actvcs = info.uq.link.activeVcs;
                info.hdr.type     = ISAR_ACTV_VC;                   /* now find the open connections */
                printf("ISAR_ACTV_VC %d\n", adap);
                for (vc = 0; vc < actvcs; ++vc) {
                        info->uq.adapter.adapter = vc;
                        if ((err = isar_query(&info)) != IA_SUCCESS)
                          return;
                        printf("  %5d: vcid=0x%08x vpi=%-4d vci=%-5d slot=%-3d state=0x%-2x tqt=%d\n",
                          vc, info.uq.vcinfo.vcid, info.uq.vcinfo.vpi, info.uq.vcinfo.vci,
                          info.uq.vcinfo.slot, info.uq.vcinfo.state, info.uq.vcinfo.tqdepth);
                }
        }
```

}

Port Information

A typical print output for this API would be:

```
Port Information
bd#    : 0
port   : 0
type   : 119
speed  : 0 Mbits/sec
conn   : 1 Mbits/sec
```

```
static void
get_port_info(u32 adap,  u32 bdlink) {
        u32 err;
        isar_query_t info;

        info.hdr.status   = NULL_STATUS;
        info.hdr.link     = bdlink;
        info.hdr.bytesIn  = sizeof(info);
        info.hdr.bytesOut = sizeof(info);
        info.hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        info.hdr.type     = ISAR_PORT_INFO;
        err = isar_query(&info);
        if((err == ISAR_PASS) && (info.hdr.status == IA_ERR_NONE)) {
                sprintf(stdout, "Port Information\n");
                sprintf(stdout, "bd#     : %d\n",info.uq.port.adapter);
                sprintf(stdout, "port    : %d\n",info.uq.port.port);
                sprintf(stdout, "type    : %d\n",info.uq.port.type);
                sprintf(stdout, "speed   : %d\n",info.uq.port.speed);
                sprintf(stdout, "conn    : %d\n",info.uq.port.connector);
        }
}
```

VC Statistics

A typical print output for this API would be:

```
VPI 0 VCI 1 Statistics
aal0In    : 0
aal0Out : 0
PacketIn    : 2
PacketOut : 2
BytesIn     : 1024
BytesOut  : 1024
OamIn      : 0
OamOut   : 0
Discards   : 0
```

```
static void
get_vc_stats(u32 vcid,  u32 vpi, u32 vci, u32 bdlink) {
        u32 err;
        isar_query_t info;
        info.hdr.status   = NULL_STATUS;
        info.hdr.link     = bdlink;
        info.hdr.bytesIn  = sizeof(info);
        info.hdr.bytesOut = sizeof(info);
        info.hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        info.hdr.type     = ISAR_VC_STATS;
        info.uq.vstats.vcid       = (ISAR_VCID)vcid;
        err = isar_query(&info);
        if((err == ISAR_PASS) && (info.hdr.status == IA_ERR_NONE)) {
                sprintf(stdout, "VPI %d VCI %d  Statistics\n", vpi , vci);
                sprintf(stdout, "aal0In   : 0x%x\n", (u32)info.uq.vstats.aal0In);
                sprintf(stdout, "aal0Out  : 0x%x\n", (u32)info.uq.vstats.aal0Out);
                sprintf(stdout, "pcktsIn  : 0x%x\n", (u32)info.uq.vstats.pcktsIn);
                sprintf(stdout, "pcktsOut : 0x%x\n", (u32)info.uq.vstats.pcktsOut);
                sprintf(stdout, "octetsIn : 0x%x\n", (u32)info.uq.vstats.octetsIn);
                sprintf(stdout, "octetsOut: 0x%x\n", (u32)info.uq.vstats.octetsOut);
                sprintf(stdout, "oamIn    : 0x%x\n", (u32)info.uq.vstats.oamIn);
                sprintf(stdout, "oamOut   : 0x%x\n", (u32)info.uq.vstats.oamOut);
                sprintf(stdout, "discards : 0x%x\n", (u32)info.uq.vstats.discards);
        }
}
```

Link Statistics

A typical print output for this API would be:

```
Link Statistics
aal0In    : 0
aal0Out : 0
PacketIn    : 2
PacketOut : 2
BytesIn     : 1024
BytesOut  : 1024
OamIn      : 0
OamOut   : 0
Discards   : 0
Lost         : 0
```

```
void get_link_stats(u32 adap,  u32 bdlink) {
        u32 err;
        isar_query_t info;
        info.hdr.status   = NULL_STATUS;
        info.hdr.link     = bdlink;
        info.hdr.bytesIn  = sizeof(info);
        info.hdr.bytesOut = sizeof(info);
        info.hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        info.hdr.type     = ISAR_LINK_STATS;
        err = isar_query(&info);
        if((err == ISAR_PASS) && (info.hdr.status == IA_ERR_NONE)) {
                sprintf(stdout, "Link Statistics\n");
                sprintf(stdout, "aal0In   : 0x%x\n", (u32)info.uq.lstats.aal0In);
                sprintf(stdout, "aal0Out  : 0x%x\n", (u32)info.uq.lstats.aal0Out);
                sprintf(stdout, "pcktsIn  : 0x%x\n", (u32)info.uq.lstats.pcktsIn);
                sprintf(stdout, "pcktsOut : 0x%x\n", (u32)info.uq.lstats.pcktsOut);
                sprintf(stdout, "octetsIn : 0x%x\n", (u32)info.uq.lstats.octetsIn);
                sprintf(stdout, "octetsOut: 0x%x\n", (u32)info.uq.lstats.octetsOut);
                sprintf(stdout, "oamIn    : 0x%x\n", (u32)info.uq.lstats.oamIn);
                sprintf(stdout, "oamOut   : 0x%x\n", (u32)info.uq.lstats.oamOut);
                sprintf(stdout, "discards : 0x%x\n", (u32)info.uq.lstats.discards);
                sprintf(stdout, "lostCount : 0x%x\n", (u32)info.uq.lstats.lostCount);
        }
}
```

Control Interface

Control structure type:

```
typedef struct isar_ctrl_s {
   isar_hdr_t         hdr;                /* request header              */
   union {
      struct ctrl_init_s {
         U32         mtu;                /* in - MTU (optional)         */
         U32         seg_vccs;           /* in - seg VCC table entries  */
         U32         seg_qsize;          /* in - seg queue(s) size      */
         U32         seg_mcr;            /* in - qos granularity        */
         U32         rsm_qsize;          /* in - rsm queue(s) size      */
         U32         rate_tbl_size;      /* in - rate table size determines traffic shaping resolution */
         U32         master_rate;        /* in - sets the maximum rate of all traffic from the card */
         U32         snoop;              /* in - VxWorks snoop enable   */
         void       *bsp;                /* in - VxWorks BSP            */
         U32         entries;            /* in - VCI/VPI entries        */
         vcisvpi_t   entry[1];           /* in - entry 0 - n            */
      } init;

      struct ctrl_rate_s {
         U32         queue;              /* queue designator (0-15)     */
         U32         mode;               /* UBR,VBR,VBR-RT,CBR,ABR      */
         union {
            struct rate_ubr_s {
               U32 pcr;                  /* peak cell rate              */
            } ubr;
            struct rate_vbr_s {
               U32 pcr;                  /* peak cell rate              */
            } vbr;
         } qos;
      } rate;

      struct ctrl_vc_s {
         ISAR_VCID   vcid;               /* in - VCID handle            */
      } vc_enable;

      struct ctrl_alarm_s {
         U32         mask;               /* in - alarm mask             */
      } alarm_mask;

   } uc;
} isar_ctrl_t;
```

Initialize the adapter

The initialization function sets the driver and adapter up for operation. The parameters involved determine which VPI's will be available for opening from the application and the number of VCI's available for each of the VPI's, the maximum frame size and depth of the segmentation and reassembly queues.
The system designer can either hard code the values in the application, parse a configuration file or prompt the user for input. The information is then placed in the control structure and sent down to the API.
The configuration file is constructed in a <keyword> <value1> <value2> style for the example below.
Valid keywords:
Link - link number
MTU - maximum cpcs-pdu size
SegVccs - number of segmentation Vccs, (64-65536 depending on memory configuration of the card
SegQsize - size of the segmentation queue (64, 256, 1024 or 4096)
RsmQsize - size of the reassmbly queue (64, 256, 1024 or 4096)
SegMCR - master rate of the card in cells per second, if 0 or no entry default is 155mbps
RateTblSize - rate table size sets the resolution for traffic shaping parameters used in the vc open function
100 - 1,360 Kb/sec
256 - 531 Kb/sec
1024 - 132 Kb/sec
2048 - 66 Kb/sec

```
static u32
do_config()
{
        isar_ctrl_t *ictrl;
        static FILE *cfile = NULL;
        static char cfile_name[256] = { "/export/home/test/isarconf.0" };
        char    line[256];
        char    word1[256];
        u32 bdlink;
        int val1;
        int val2;
        int cnt;

        if ((cfile = fopen( &cfile_name[0], "r" )) == NULL) {
                fprintf (stdout, "testtool: Failed to open %s\n", cfile_name);
                return;
        }
        /* set default parameters */
        ictrl = (isar_ctrl_t *)malloc(ictrlsizeof(isar_ctrl_t) + (sizeof(vcisvpi_t) * VCISVPI_MAX)));
        if(!ictrl) {
                return (ISAR_FAIL);
        }
        ictrl->hdr.status   = NULL_STATUS;
        ictrl->hdr.link     = bdlink;
        ictrl->hdr.bytesIn  = sizeof(isar_ctrl_t);
        ictrl->hdr.bytesOut = sizeof(isar_ctrl_t);
        ictrl->hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;

        ictrl->uc.init.mtu = MAX_MTU;
        ictrl->uc.init.seg_vccs = 4096;
        ictrl->uc.init.seg_qsize = 64;
```

```
ictrl->uc.init.rsm_qsize = 64;
ictrl->uc.init.seg_mcr = 0;
ictrl->uc.init.rate_tbl_size = 2048;
ictrl->uc.init.snoop = 0;
ictrl->uc.init.bsp = NULL;
ictrl->uc.init.entries = 0;

while (fgets(line, 256, cfile) != NULL) {
        cnt = sscanf (line, "%s%d%d", word1, &val1, &val2);

        if ((cnt == EOF) || (cnt == 0) || (word1[0] == '#')) {
                continue;
        }
        if (strcmp (word1, "Link") == 0) {
                bdlink = val1;
        }
        if (strcmp (word1, "MTU") == 0) {
                ictrl->uc.init.mtu = val1;
        }
        if (strcmp (word1, "SegVccs") == 0) {
                ictrl->uc.init.seg_vccs = val1;
        }
        if (strcmp (word1, "SegQsize") == 0) {
                ictrl->uc.init.seg_qsize = val1;
        }
        if (strcmp (word1, "SegMCR") == 0) {
                ictrl->uc.init.seg_mcr = val1;
        }
        if (strcmp (word1, "RateTblSize") == 0) {
                ictrl->uc.init.rate_tbl_size = val1;
        }
        if (strcmp (word1, "RsmQsize") == 0) {
                ictrl->uc.init.rsm_qsize = val1;
        }
        if (strcmp (word1, "VPI") == 0) {
                ictrl->uc.init.entry[ictrl->uc.init.entries].vpi = val1;
                ictrl->uc.init.entry[ictrl->uc.init.entries].vci = val2;
                ictrl->uc.init.entries++;
        }
}
fclose(cfile);

 sprintf(stdout, "Current Initialization Parameters:\n");
sprintf(stdout, "MTU     : %d\n", ictrl->uc.init.mtu);
sprintf(stdout, "SegVccs : %d\n", ictrl->uc.init.seg_vccs);
sprintf(stdout, "SegQsize: %d\n", ictrl->uc.init.seg_qsize);
sprintf(stdout, "SegMCR  : %d\n", ictrl->uc.init.seg_mcr);
sprintf(stdout, "RsmQsize: %d\n", ictrl->uc.init.rsm_qsize);
sprintf(stdout, "VccEntry: %d\n", ictrl->uc.init.entries);
for(x = 0; x < ictrl->uc.init.entries; x++) {
        sprintf(stdout, "vpi %d vci count %d\n", ictrl->uc.init.entry[x].vpi, ictrl->uc.init.entry[x].vci);
```

```
        }
        ictrl->hdr.type = ISAR_INIT;
        ictrl->hdr.bytesIn  = sizeof(isar_ctrl_t) + (sizeof(vcisvpi_t) * (ictrl->uc.init.entries - 1));
       ictrl->hdr.bytesOut = ictrl->hdr.bytesIn;
       err = isar_control(ictrl);
        if ((err != ISAR_PASS) || (ictrl->hdr.status != IA_ERR_NONE)) {
                if(ictrl->hdr.status == IA_ERR_INITED) {
                        sprintf(stdout, "Driver is already configured, you must reboot to use new parameters\n");
                }
                sprintf(stdout, "isar_control failed...type %d err 0x%x status %d\n",
                        ictrl->hdr.type, err, ictrl->hdr.status);
        }
}
```

Setting Alarms

Alarms are available for LOS (loss of signal, cable pull) or OAM cell reception. The alarm funtion operates on a mask value and can be enabled or disabled at anytime. When an alarm condition is encountered the supplied alarm function is called from the API. The following source example is for setting up an alarm for LOS.

```
static u32
setup_LOS_alarm()
{
        isar_ctrl_t *ictrl;
        isar_alarm_t los_alarm;
        u32 err;

        ictrl = (isar_ctrl_t *)malloc(sizeof(isar_ctrl_t));
        if(!ictrl) {
                return(ISAR_FAIL);
        }
        ictrl->hdr.status   = NULL_STATUS;
        ictrl->hdr.link     = bdlink;
        ictrl->hdr.bytesIn  = sizeof(isar_ctrl_t);
        ictrl->hdr.bytesOut = sizeof(isar_ctrl_t);
        ictrl->hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        ictrl->hdr.type = ISAR_ALARM_MASK;
        ictrl->uc.alarm_mask.mask = (ISAR_ALARM_PHY_LOS);
        err = isar_control(ictrl);
        if((err != ISAR_PASS) || (ictrl->hdr.status != IA_ERR_NONE)) {
                sprintf(stdout, "isar_control ALARM_MASK failed...err 0x%x status 0x%x\n",
                err, ictrl->hdr.status);
                return(ISAR_FAIL);
        } else {
                los_alarm.alarms = ISAR_ALARM_PHY_LOS;
                los_alarm.mode = ISAR_ALARM_EDGE;
                los_alarm.arg = 0;
                los_alarm.ind = &do_alarm;
                los_alarm.tag = LOS_ALARM_TAG;
                isar_alarms(&los_alarm);
                ictrl->hdr.type = ISAR_ALARM_ENABLE;
                err = isar_control(ictrl);
                if((err != ISAR_PASS) ||  (ictrl->hdr.status != IA_ERR_NONE)) {
                   sprintf(stdout, "isar_control ISAR_ALARM_ENABLE failed...  err 0x%x status 0x%x\n",
                        err, ictrl->hdr.status);
                }
        }
}

static void
do_alarm(void *arg))
{
        isar_alarm_ind_t *los_ind = (isar_alarm_ind_t *)arg;
        sprintf(stdout, "LOS Alarm recieved\n");
        sprintf(stdout, "alarms: 0x%x\n",los_ind->alarms);
        sprintf(stdout, "arg   : 0x%x\n",(u32)los_ind->arg);
```

```
        sprintf(stdout, "link  : 0x%x\n",los_ind->link);
        sprintf(stdout, "vpi   : 0x%x\n",los_ind->vpi);
        sprintf(stdout, "vci   : 0x%x\n",los_ind->vci);
        sprintf(stdout, "vcid  : 0x%x\n",(u32)los_ind->vcid); /* for OAM cell alarms */
}
```

Setting Loopback Mode

The loopback mode API is for diagnostic purposes. The default mode of the driver is to have loopback disabled.. The following source example is for enabling loopback mode.

```
static u32
setup_loopback_mode()
{
        isar_ctrl_t *ictrl;
        isar_alarm_t los_alarm;
        u32 err;

        ictrl = (isar_ctrl_t *)malloc(sizeof(isar_ctrl_t));
        if(!ictrl) {
                return(ISAR_FAIL);
        }
        ictrl->hdr.status   = NULL_STATUS;
        ictrl->hdr.link     = bdlink;
        ictrl->hdr.bytesIn  = sizeof(isar_ctrl_t);
        ictrl->hdr.bytesOut = sizeof(isar_ctrl_t);
        ictrl->hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        ictrl->hdr.type = ISAR_LPBK_ENABLE;
        err = isar_control(ictrl);
        if((err != ISAR_PASS) ||  (ictrl->hdr.status != IA_ERR_NONE)) {
                sprintf(stdout, "isar_control ISAR_LPBK_ENABLE failed...  err 0x%x status 0x%x\n",
                err, ictrl->hdr.status);
        }
}
```

Setting Trace Mode

The trace mode API is for diagnostic purposes. The default mode of the driver is to have trace disabled.. When enabled, trace mode will cause the API to display data to stdout and the driver to send data to the console. The following source example is for enabling trace mode.

```
static u32
setup_trace_mode()
{
        isar_ctrl_t *ictrl;
        isar_alarm_t los_alarm;
        u32 err;

        ictrl = (isar_ctrl_t *)malloc(sizeof(isar_ctrl_t));
        if(!ictrl) {
                return(ISAR_FAIL);
        }
        ictrl->hdr.status   = NULL_STATUS;
        ictrl->hdr.link     = bdlink;
        ictrl->hdr.bytesIn  = sizeof(isar_ctrl_t);
        ictrl->hdr.bytesOut = sizeof(isar_ctrl_t);
        ictrl->hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        ictrl->hdr.type = ISAR_TRACE_ENABLE;
        err = isar_control(ictrl);
        if((err != ISAR_PASS) ||  (ictrl->hdr.status != IA_ERR_NONE)) {
                sprintf(stdout, "isar_control ISAR_TRACE_ENABLE failed...  err 0x%x status 0x%x\n",
                err, ictrl->hdr.status);
        }
}
```

Setting VC Mode

The VC mode API is for controlling traffic flow. The default mode of the driver is to have a VC enabled. when opened. After a connection is opened the application can enable/disale the connection at anytime. This function is usefull to stop the API from calling the recieve functions callback routine.The following source example is for controlling a VC.

```
static u32
do_vc_control(u32 mode, u32 vcid)
{
        isar_ctrl_t *ictrl;
        isar_alarm_t los_alarm;
        u32 err;

        ictrl = (isar_ctrl_t *)malloc(sizeof(isar_ctrl_t));
        if(!ictrl) {
                return(ISAR_FAIL);
        }
        ictrl->hdr.status   = NULL_STATUS;
        ictrl->hdr.link     = bdlink;
        ictrl->hdr.bytesIn  = sizeof(isar_ctrl_t);
        ictrl->hdr.bytesOut = sizeof(isar_ctrl_t);
        ictrl->hdr.tag      = (ISAR_USR_TAG)ISAR_HDR_TAG;
        ictrl->uc.vc_enable.vcid = (ISAR_VCID)vcid;
        if(mode) {
                ictrl->hdr.type = ISAR_VC_ENABLE;
        } else {
                ictrl->hdr.type = ISAR_VC_DISABLE;
        }
        err = isar_control(ictrl);
        if((err != ISAR_PASS) ||  (ictrl->hdr.status != IA_ERR_NONE)) {
                sprintf(stdout, "isar_control ISAR_VC_DISABLE failed...  err 0x%x status 0x%x\n",
                err, ictrl->hdr.status);
        }
}
```

Open and Close a connection

In order for an application to transmit or recieve data it must first open a connection. The user can then control the flow of that connection using the vc enable/disable API.

The open structure provides entries for QOS parameters. For CBR and VBR connection modes the RateTblSize set during the initialize process determines the possible resolution for the PCR value.

Open Connection

Open connection structure type:

```
typedef struct isar_vc_open_s {
        U32             status;                 /* completion status/error code */
        U32             link;                   /* if/port number               */
        U32             vci;                    /* virtual curcuit identifier   */
        U32             vpi;                    /* virtual path identifier      */
        U32             type;                   /* AAL0,AAL5                    */
        U32             mode;                   /* UBR,VBR,ABR,CBR              */
        U32             pcr;                    /* PCR, UBR, VBR, CBR           */
        U32             scr;                    /* SCR, VBR only        */
        U32             mbs;                    /* MBS, VBR only        */
        ISAR_RCV_IND    *rcv_ind;               /* receive indication callback  */
        ISAR_IND_ARG    rcv_arg;                /* opaque receive argument      */
        ISAR_USR_TAG    tag;                    /* opaque user tag/id           */
        U32             txq_depth;              /* max tx's on TxQ at one time  */
        char            addr[VCC_ADR_LEN];      /* address                      */
        U32             enable;                 /* enable immediately (t/f)     */
} isar_vc_open_t;
```

The txq_depth variable is intended for use with slow rate VC's. This variable allows the driver to throttle traffic on slow rate VC's and not let the slow rate VC consume all of the resources. For example if a 300Kb/sec rate VC was transmitting it will take over 5 seconds for the hardware to segment that frame. In the meantime subsequent transmit requests for this same VC would be backlogged on the controller.

```
static u32
do_open(u32 vpi, u32 vci, u32 type, u32 mode, u32 pcr, u32 scr, u32 mbs)
{
        ISAR_VCID       vcid;
        isar_vc_open_t  vopen;
        u32 rateq_num;
        vopen.status       = NULL_STATUS;
        vopen.link         = bdlink;
        vopen.vci          = vci;
        vopen.vpi          = vpi;
        vopen.type         =  type; /* ISAR_AAL0 or ISAR_AAL5 */
        vopen.mode         =  mode; /* ISAR_UBR, ISAR_CBR or ISAR_VBR */
        vopen.pcr          = pcr; /*0 = set for full line rate, used for all modes */
        vopen.scr         = scr; /* vbr only parameter */
        vopen.mbs            = mbs; /* vbr only parameter */
        vopen.enable      = ISAR_TRUE;
        vopen.rcv_ind     = &do_receive;
        vopen.rcv_arg     = (ISAR_IND_ARG)((vpi << 16) | (vci & 0xffff));
```

```
        vopen.tag           = VCOPEN_USR_TAG;
        vopen.txq_depth = 0;  /* no limit on txq depth */

        vcid = isar_vc_open(&vopen);

        /* check for STATUS error */
        if (vopen.status == NULL_STATUS){
                sprintf(stdout, "vcopen did not return any status\n");
                 vcid = NULL;
        }
         if (vopen.status != IA_ERR_NONE) {
                sprintf(stdout, "vcopen returned  error %d\n", vopen.status);
                vcid = NULL;
        }
        return((u32)vcid);
}
```

Close Connection

When the application is no longer interested in a given connection it can be closed. The API should insure that all open connections are closed before exiting the application.

```
static u32
do_close(u32 vcid)
{
        u32 err;
        err = isar_vc_close((ISAR_VCID)vcid);
        if(err != ISAR_PASS) {
                sprintf(stdout, “isar_vc_close failed with %d for vcid 0x%x\n”, err, vcid);
                return(ISAR_FAIL);
        }
        return(ISAR_PASS);
}
```

Transmit and Recieve Data

The transmit and receive data paths for the 4576 do not use the standard streams interface of Solaris. Instead all data traffic is passed through the API.

Transmit

If the transmit function fails due to IA_ERR_LIST it is a result of all the drivers resources having been consumed by previous transmit requests. (all segmentation buffers in use) This could happen very easily on very slow rate VC's. The example below simply retries the operation until it succeeds. However, it is suggested that in the event of an IA_ERR_LIST return that a backoff algorithm be invoked.

```
extern u32 host_tx_tbd_fail;
static u32
do_transmit(u32 vcid, void *buf,  u32 len)
{
        isar_vc_send_t ist;

        ist.vcid = NULL;
        ist.ctrl = 0;
        ist.length = len;
        ist.datap = buf;
        ist.tag = (ISAR_USR_TAG)ISAR_HDR_TAG;
        ist.vcid = (ISAR_VCID)vcid;
         do {
                err = isar_vc_send(&ist);
                if(err  != ISAR_PASS) {
                        if(err == IA_ERR_LIST) {
                                host_tx_tbd_fail++;
                        } else {
                                sprintf(stdout, "do_tx failure %d...exiting\n", cc);
                                return(ISAR_FAIL);
                        }
                }
        } while (err != ISAR_PASS);
        return(ISAR_PASS);
}
```

Receive

The recieve function is registered with the API when the connection is opened. If the connection is enabled, when a packet arrives the receive callback function will be invoked by the API. The application is responsible for freeing the buffer passed up by the API. (MEM_FREE is the library function to use)

```
extern u32 mtu;
static void
do_receive(void *arg)
{
        isar_rcv_ind_t  *rcv = (isar_rcv_ind_t *)arg;
        sprintf(stdout, "RX vcid 0x%x  len %d\n", (u32)rcv->vcid,  rcv->length);
        MEM_FREE(rcv->datap, mtu);
}
```