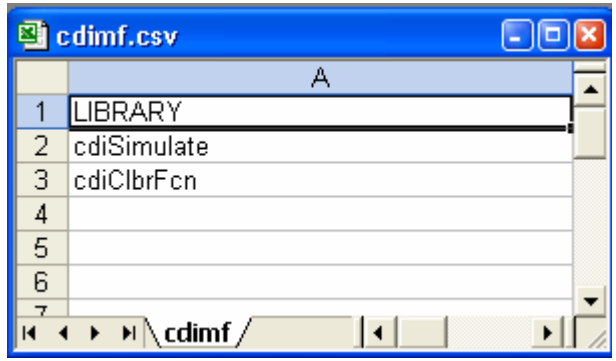
CDI Bus Plugs

How to write a simple one …

How CDI works
(i.e. what's a bug plug?)

- When CDI loads it looks for a manifest file (cdimf.csv) :
- Simplest case : single "LIBRARY" column :
- Entries refer to "bus plug" libraries which should be loaded for "this" machine.
- Windows appends a ".dll", Unix appends a ".so" and loads the library.
- In this example: cdiSimulate.dll is loaded, followed by cdiClbrFcn.dll (on a windows machine).



How a Bus Plug works

- In order to work with CDI, the bus plug must register at least 3 functions with CDI:
  - cdiRegisterBusInitialization();
    - Registers a function which handles bus (plug) initialization.
  - cdiRegisterBusHandler();
    - Registers a function with handles bus (plug) read/write requests (can be reentrant/state dependent).
  - cdiRegisterBusCleanup();
    - Registers a function that frees bus (plug) resources upon exit.

How a Bus Plug works (simulation example)

Library's prolog code calls the bus functions :

Windows prolog :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdarg.h>
#include <math.h>
#include <errno.h>
#include "cdi.h"
#include "tinecdi.h"
#include "cdiSimulateLib.h"

#define BUSPLUG_NAME "SIMULATE"
int cdiInitSimulate(void)
{
  int cc = 0;

  if ((cc=cdiRegisterBusFilter(BUSPLUG_NAME,filterSimulation)) != 0 ) goto err;
  if ((cc=cdiRegisterBusInitialization(BUSPLUG_NAME,initSimulation)) != 0 ) goto err;
  if ((cc=cdiRegisterBusHandler(BUSPLUG_NAME,simulationHandler)) != 0 ) goto err;
  if ((cc=cdiRegisterBusCleanup(BUSPLUG_NAME,exitSimulation)) != 0 ) goto err;
  if ((cc=cdiRegisterBusScanner(BUSPLUG_NAME,scanSimulationBus)) != 0 ) goto err;
err:
  if (cc)
  {
    cdilog("%s : failure in bus registration SEDPC",erlst[cc]);
  }
  return cc;
}

BOOL WINAPI DllMain(HINSTANCE hInstDll,DWORD fdwReason,LPVOID lpvReserved)
{
  switch (fdwReason)
  {
    case DLL_PROCESS_ATTACH:  // DLL being loaded
      cdiInitSimulate();
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
    case DLL_PROCESS_DETACH:  // DLL being unloaded
      break;
  }
  return (TRUE);
}
```

How a Bus Plug works (simulation example)

Unix prolog :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/kd.h>
#include <sys/time.h>
#include <dlfnc.h>
#include "tine.h"
#include "cdi.h"
#include "cdiSimulateLib.h"

#define BUSPLUG_NAME "SIMULATE"

__attribute__((constructor)) void cdiInitSimulate(void)
{
    int cc = 0;

    if ((cc=cdiRegisterBusFilter(BUSPLUG_NAME,filterSimulation)) != 0 ) goto err;
    if ((cc=cdiRegisterBusInitialization(BUSPLUG_NAME,initSimulation)) != 0 ) goto err;
    if ((cc=cdiRegisterBusHandler(BUSPLUG_NAME,simulationHandler)) != 0 ) goto err;
    if ((cc=cdiRegisterBusCleanup(BUSPLUG_NAME,exitSimulation)) != 0 ) goto err;
    if ((cc=cdiRegisterBusScanner(BUSPLUG_NAME,scanSimulationBus)) != 0 ) goto err;

err:
    if (cc)
    {
      cdilog("%s : failure in bus registration SEDAC",erlst[cc]);
    }
    return;
}
```

How a Bus Plug works
(simulation initialization)

```
__declspec(dllexport) int initSimulation(int busLine, int cdiLine, int numberDevices, char *parameterList)
{
  cdi_mutex_t tmpMutex;
  int lastError, error = 0;
  /* try to get Mutex if this is the first application  */
  lastError = cdiMutexInit(&tmpMutex, cdiLine);
  if ( lastError == ERROR_ALREADY_EXISTS ) cdiMutexLock(&tmpMutex);
  error = initHardware(busLine);
  if ( lastError == ERROR_ALREADY_EXISTS ) cdiMutexUnlock(&tmpMutex);
  return error;
}
```

How a Bus Plug works
(simulation exit)

```
__declspec(dllexport) int exitSimulation(int ln)
{
  return 0;
}
```

How a Bus Plug works (simulation handler)

```
__declspec(dllexport) void simulationHandler(CdiRequestInfoBlk *pReq)
{
  int access, length = sizeof(short); /* fix this to read 1 word from the bus */
  void *pData;

  /* Note:
     pReq->active can contain 'state' information in case the handler needs to be re-entered
     We will not make use of it here in this simple simulation example
  */
  switch (pReq->accessFlag)
  {
    default: /* there are more complicated cases, but let s keep it simple here */
    case cdiReadFlag:
      access = CA_READ;
      /* a 'READ' is regarded as input into the bus plug */
      pData = (void *)pReq->pInData;
      break;
    case cdiWriteFlag:
      access = CA_WRITE;
      /* a 'WRITE' is regarded as output from the bus plug */
      pData = (void *)pReq->pOutData;
      break;
  }

  cdiMutexLock(&((CdiLineInfoBlk *)pReq->pDev->pcdiLine)->lineMutex);

  pReq->errorCode = simulateBusAction(
          (WORD)((CdiLineInfoBlk *)pReq->pDev->pcdiLine)->busLine,
          (WORD)pReq->pDev->cdiAddr,
          (WORD)(pReq->pDev->cdiSubAddr + pReq->devParameters[0]),
          (int)access,
          (WORD *)pData,
          (int)length / sizeof(short) );

  if (pReq->errorCode != 0)
  { /* bus action not successful */
    if (pReq->perror != NULL) *(pReq->perror) = pReq->errorCode;
    pReq->deviceError = pReq->errorCode;
    pReq->pDev->numberDeviceError++;
  }

  pReq->active = finishRequest; /* must signal that the request is finished */
  cdiMutexUnlock(&((CdiLineInfoBlk *)pReq->pDev->pcdiLine)->lineMutex);
  return;
}
```

How a Bus Plug works (CdiRequestInfoBlk)

```
typedef struct
{
  int active;      /* request process steps */
  int callBackIndex;  /* can be user signed (< 0x1000) or requestNumber + 0x1000 */
  CdiDeviceInfoBlk *pDev; /* referenced device */
  void *pInData;     /* from device */
  unsigned int inLength;
  void *pOutData;    /* to device */
  unsigned int outLength;
  short inType, outType;  /* char, short, int, float or double  */
  short inFormatCode, outFormatCode;  /* original caller's code */
  int hasPatternFilter;
  BYTE patternFilter[CDI_PATTERN_FILTER_SIZE]; /* was ->  void *patternFilter; */ /* compare pattern or E
  int hasDataMask;
  BYTE mask[8];  /*   was -> void *mask; */
  int maskValue;
  int devParameters[NUMBER_EXEC_PARAM];
  int timeout; /* in milliseconds: < 0 return only on time out or error; 0: wait forever */
  int registerTime; /* contract register time */
  int requestTime; /* send time */
  int pollTime;
  int requestCounter;
  short accessFlag, /* read 1, write 2, read then write 3, write then read 4 */
    conditionFlag; /* for reading data (readConditionNumeraters), bit: 0x8 for mask ON flag */
  int needsCalibration;
  int requestNumber;  /*  reqIndex + (cdiLine << 16) */
  int reqIndex;    /* own index number: 0, 1... */
  void *groupReq; /* grouped request, group true if != NULL */
  short NgroupsReqs,     /* unmber of connected requests for mainRequest, but index number for each reques
    NgroupRets,    /* number of data returnd or timeout... */
    NgrpErrors,    /* number of errors of this group  */
    groupSyncStart;     /* synchronous flag for group start */
  short dataReturnCondition, /* databack: 1; conditionOK: 2 */
    asynCallFlag;
  int deviceError;
  BYTE  cpOutData[DEF_INPUT_BYTES]; /* for default input */
  short bufLastData[4]; /* for compare  */
  int errorCode, *perror;
  int responseTime; /* last response time in ms */
  short pollFlag, /* infinite loop for async call */
    lockRequestFlag,  /* lock this request until cancel calls */
    userCallDown, /* callback or syncSem done flag */
    cancelFlag;    /* cancel callback */
  sem_t sync_SemId;
  int bufIndex;    /* for bus plug  */
  void (*callback)(int, int); /* callBackIndex, errorCode */
  int freeInDataMemoryOnCancel;
  int freeOutDataMemoryOnCancel;
} CdiRequestInfoBlk;
```

How a Bus Plug works (CdiDeviceInfoBlk)

```
/* Bind address to device name */
typedef struct
{
  char devName[CDI_DEVICE_NAME_SIZE];  /* registered (hashed) device name */
  int cdiIndex;       /* start from 0, for pDev[] */
  int devNumber;      /* start from 1, unique number */
  void *pcdiLine;     /*  pointer to CdiLineInfoBlk (before: int cdiLine; index for Cd
  int cdiAddr;        /* module address */
  int cdiSubAddr;     /* module sub-address */
  int numberDeviceError;
  int *devParameters;/* default bus parameters for this device */
  BYTE *dataMask;     /* default data mask */
  int dataMaskStartIndex;
  int dataMaskStopIndex;
  BYTE *dataPattern; /* default data tolerance/pattern */
  int dataPatternStartIndex;
  int dataPatternStopIndex;
  int dataPatternTrigger;
  BYTE *dataInput;    /* default data input for WRITE commands */
  int dataInputSize; /* maximum allowed data input size in bytes (if > 0) */
  int dataOuputSize; /* maximum allowed data output size in bytes (if > 0) */
  int dataOuput2ndSize; /* maximum allowed Seconnd data output size in bytes (if > 0)
  int dataFormat;     /* default data type */
  int reqFormat;      /* requested data type */
  int allowedAccess; /* allowed bus access flag */
  CdiRule *clbrRules;/* associated calibration rules for device readout */
  CdiRule *rvrsRules;/* associated reverse calibration rules for device sendto */
  char lngName[CDI_DEVICE_LONGNAME_SIZE];  /* associated long device name */
  char description[CDI_DEVICE_DESC_SIZE];/* associated device description */
  float maximum;      /* display maximum */
  float minimum;      /* display minimum */
  char precision[16];/* display format a la width.decimal */
  char units[16];     /* display units */
} CdiDeviceInfoBlk;
```

How a Bus Plug works (simulation project)