Using the Common Device Interface in TINE

Philip Duval and Honggong Wu

this.Contens()

- A few words about driver support
- Motivation behind CDI; Design goals
- How CDI works
- CDI manifest; CDI database
- A few server examples using CDI
- Status

hardware support (1)

- do it yourself
  - use the drivers that come with what you bought, or write your own
  - mostly sedac for HERA + pre-accelerators (+ gpib, rs232, increasing use of can, dsps ...)
  - TINE via LabView (leverage the LV drivers)
  - different interface API: bus and operating system dependent

hardware support (2)

- PETRA III
  - mixed
  - more emphasis on CAN, TwinCat, + SEDAC, GPIB, rs232, vme, siemensPLCs, Libera etc.

http://www.beckhoff.com

http://www.i-tech.si/

Application Programmers

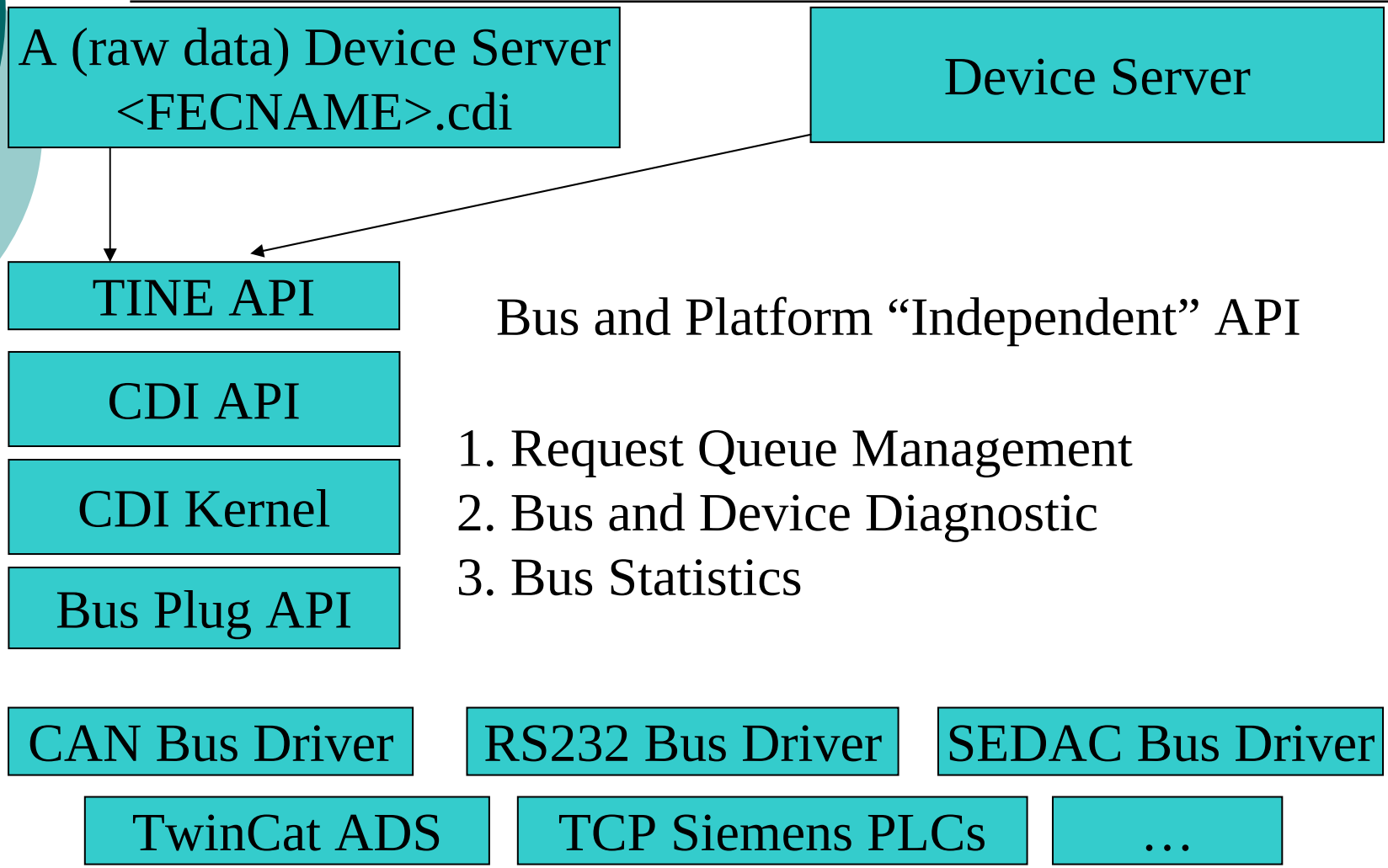
- either learn an interface API for each hardware card of a bus, and each operating system
- or create a common interface for device access

- use TINE CDI !

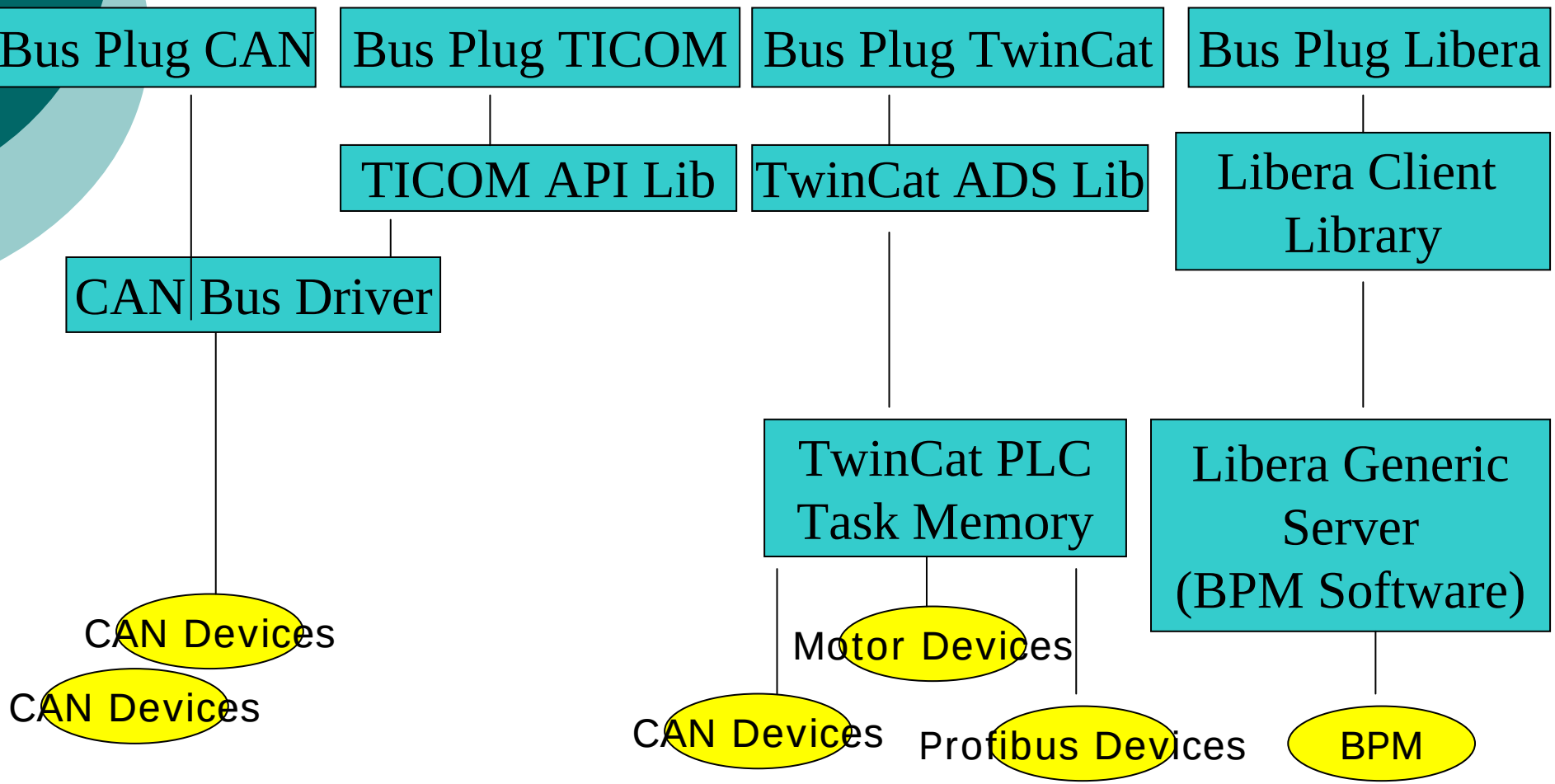
CDI API

- have a general and powerful client API in TINE
  - use it for accessing the hardware => no new API to learn!
- Allow:
  - the good old "do it yourself" method if you want to or need to
  - direct CDI API interface (TINE similar for device registration and device access)
  - direct TINE client calls for accessing hardware devices + database registration of devices.

Device Server using TINE CDI



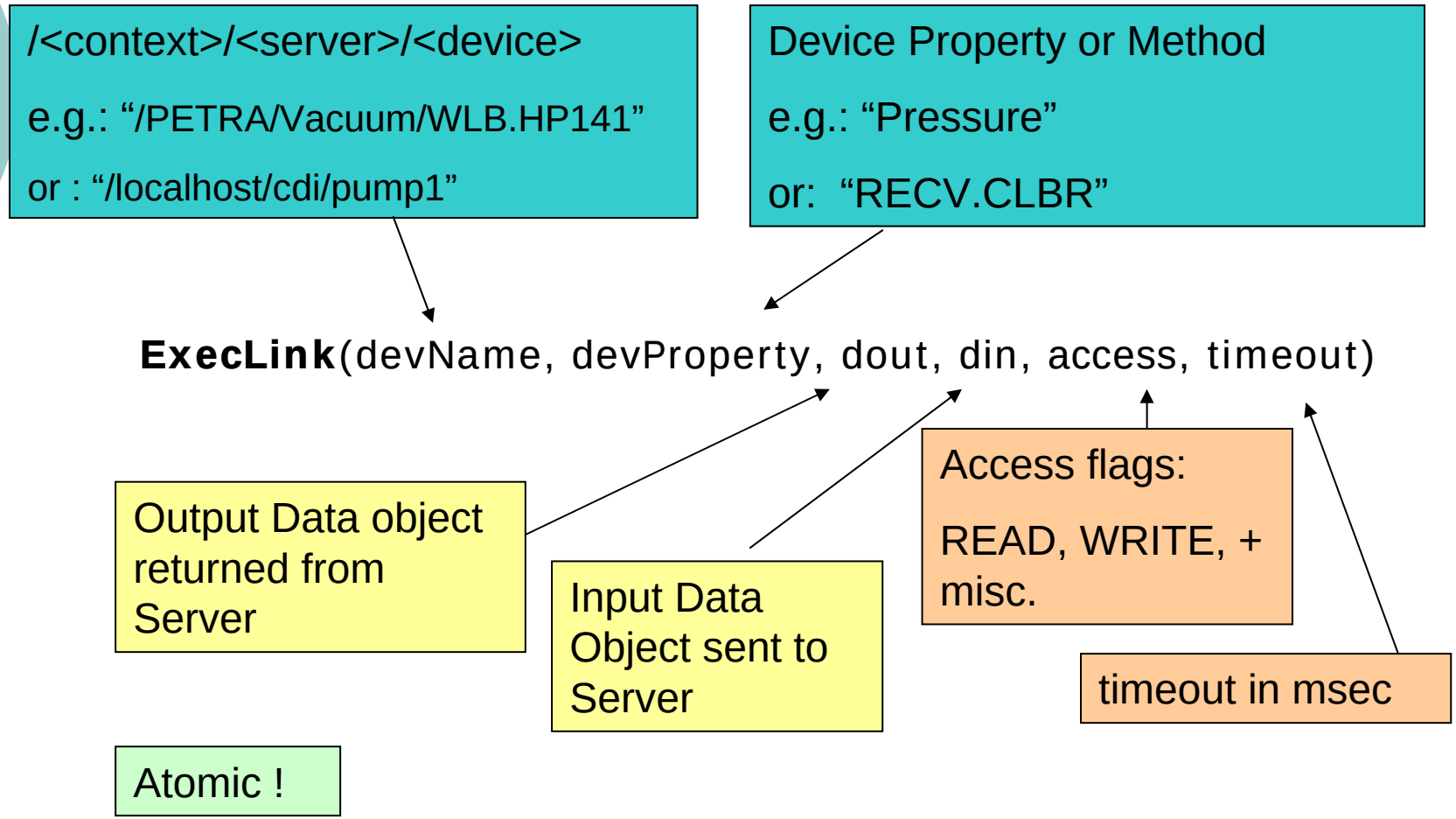
CDI Bus Plug?


Bus Plug CAN
Bus Plug TICOM
Bus Plug TwinCat
Bus Plug Libera
TICOM API Lib
TwinCat ADS Lib
Libera Client Library
CAN Bus Driver
TwinCat PLC Task Memory
Libera Generic Server (BPM Software)
CAN Devices
CAN Devices
Motor Devices
CAN Devices
Profibus Devices
BPM

CDI Bus Plug continue…

- For a new Bus type, a few functions need to be implemented, use the driver API for the target platform
  - Open Bus, initialization
  - Close Bus, cleanup
  - Read/Write Data
  - Filter for input parameters
  - Bus Scan
  - … …

Synchronous Calls



Asynchronous Calls

Analogous to synchronous parameters …

AttachLink(devName, devProperty, dout, din, access, pollrate,
void (*callback)(int,int), callbackID, mode)

Callback with callback id and status code …

CM_CANCEL
CM_SINGLE
CM_REFRESH
CM_POLL
CM_NETWORK
CM_GROUPED
CM_WAIT
+ …

Atomic !

CDI API Details

- **Device Name** (name or number):
  - “/localhost/cdi/#1”
  - “/localhost/cdi/#1-#100”
  - “/localhost/cdi/#1,#3-#10,#99”
  - “/localhost/cdi/pump1 – pump100”
  - ...
- **Device Properties** (methods)
  - “RECV”
  - “SEND”
  - “RECV.SEND.ATOM”
  - “SEND.RECV.ATOM”
  - “RECV.CLBR”
  - “SEND.RECV.CLBR”
  - “ADDR”
  - “BUSNAME”
  - “BUSSCAN”
  - “BUSERRORS”
  - ...

Can Use Device Name or Device Number !

Read/Write Raw or Calibrated data

Atomic pair-wise access

Device information

CDI: How it works …

1) Bus Manifest :

cdimf.csv

	A
1	LIBRARY
2	cdiCanEsd
3	cdiSedac
4	cdiTwinCat
5	cdiRs232
6	

← Bus Interface Plugs

2.) **cdiLoadLib**(“cdiCanEsd.dll”) - Windows
cdiLoadLib(“libcdiCanEsd.so”) - Unix
cdiLoadLib(“cdiCanEsdLib.o”) - VxWorks
Etc. …

3). Library’s prologue code ‘plugs’ into CDI:

int **cdiRegisterBus**(char *busName);
int **cdiRegisterBusInitialization**(char *busName,int (*fcn)(int,int,int,char *));
int **cdiRegisterBusHandler**(char *busName,void (*fcn)(CdiRequestInfoBlk *));
int **cdiRegisterBusCleanup**(char *busName,int (*fcn)(int));
int **cdiRegisterBusFilter**(char *busName,int (*fcn)(int *, int));
int **cdiRegisterBusScanner**(char *busName,int (*fcn)(char *,char *, int));
…

CDI: How it works …

- Either register CDI devices dynamically via API or
- Use a CDI database to register devices at initialization:

cdiaddr.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	NUMBER	NAME	BUS	LINE	INDEX	ADDRESS	MASK	PATTERN	TOLERAI	ACCESS	INPUT	FORMAT	LIMIT	RULE
2	1	SPS-WQPS	CAN:500	2	0	1:100:0:0:1:0	0xffff	0	0	WRRD	32	short	01:01	
3	2	SPS-WQPS1	CAN:100	1	0	1:100:0x600:0x3011:0:0	0xffff	0	0	READ		long		
4	3	SerialDevice1	RS232:9600:0:1:1:0:0:0	2	1	0	0xffff	0	0	READ		text		
5	4	Temperature1	SEDAC	1	0	16.252:0	0xffff	0	0	WRRD	8	short	01:01	"+-120:*0.01:+-100.0:*52:^.5"
6	5	Temperature2	SEDAC	1	1	16.253:0	0xffff	0	0	WRRD	12	short	01:01	"+-120:*0.01:+-100.0:*52:^.5"

Device number

Device name

Bus + parameters

Device Address

Data manipulation

Device Access

Data Access Rules

Data Calibration Rules

CDI Initialization Flow …

- Application calls cdiInitialize()
  - cdi library loads and reads manifest
    - each entry in manifest references a bus plug library
    - cdi calls cdiLoadLibrary() for each bus plug in the manifest
      - each bus plug loads()
      - bus plugs register all bus handlers with CDI
  - cdi looks for local database
    - Reads database
    - Registers individual device information
- Application can then make calls to CDI
  - via TINE client interface (*ExecLink*())
  - via CDI interface (TINE similar) (*cdiExecLink*())

CDI Examples

```
dout.dArrayLength = 100;
dout.dFormat = CF_INTINT;
dout.data.vptr = rbPressData; // <- INTINT[100] object
AttachLink("/localhost/cdi/#1-#100", "RECV.CLBR", &dout,
    NULL,1000, cbPressData)
```

Reads devices 1 to 100, calibrates the raw data, fills in rbPressData[] and calls the callback cbPressData() at 1 Hz. (value + status pairs)

```
dout.dArrayLength = 100;
dout.dFormat = CF_UINT16;
dout.data.sptr = rbPressData;
AttachLink("/localhost/cdi/#1", "RECV.CLBR", &dout, NULL,1000,
cbPressData)
```

Reads device 1 100 times, calibrates the raw data, fills in rbPressData[] and calls the callback cbPressData() at 1 Hz.

CDI Examples

```
din.dArrayLength = 1;
din.dFormat = CF_UINT16;
din.data.sptr = &setValue;
ExecLink (“/localhost/cdi/#16”, ”SEND”, NULL, &din,1000)
```

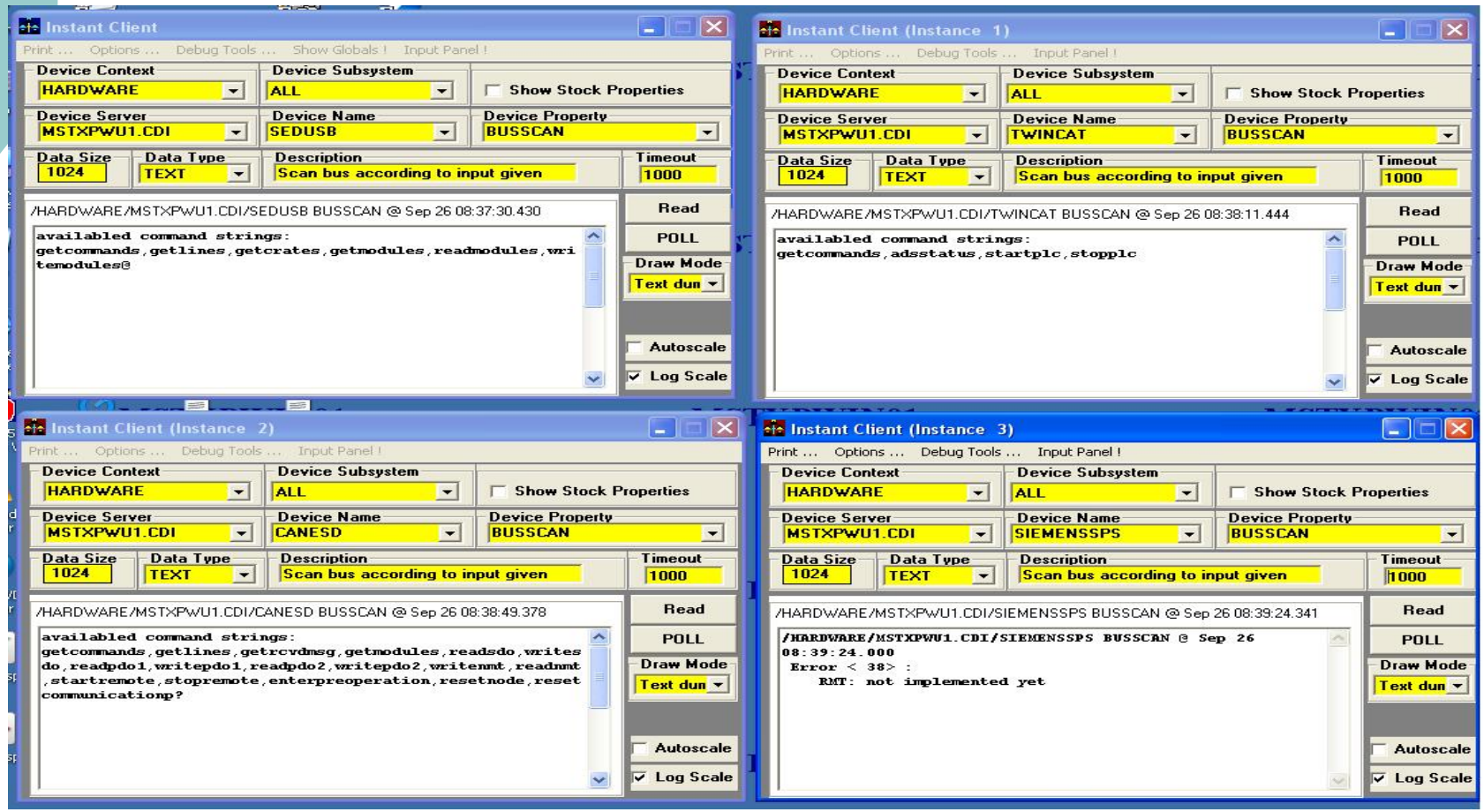
Sends ‘setValue’ to device #16

```
din.dArrayLength = 1;
din.dFormat = CF_UINT16;
din.data.sptr = &setValue;
dout.dArrayLength = 1;
dout.dFormat = CF_UINT16;
dout.data.sptr = &rbValue;
ExecLink (“/localhost/cdi/#1”, ”SEND.RECV.ATOM”, &dout, &din,1000)
```

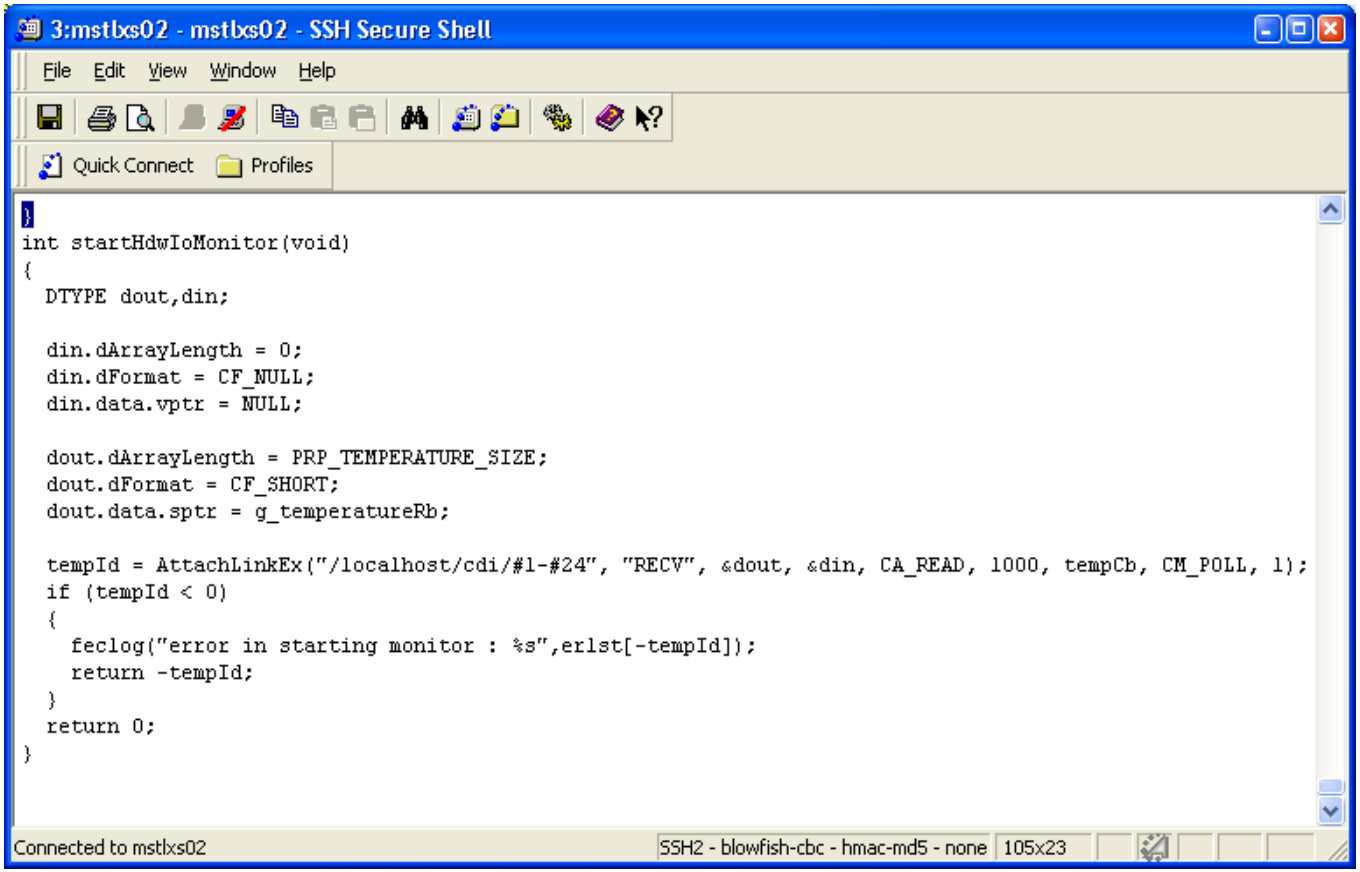
Sends ‘setValue’ to device #1 and reads rbValue from device #1 atomically.

BusScan Implementation Examples

- Sedac, TwinCat and CAN



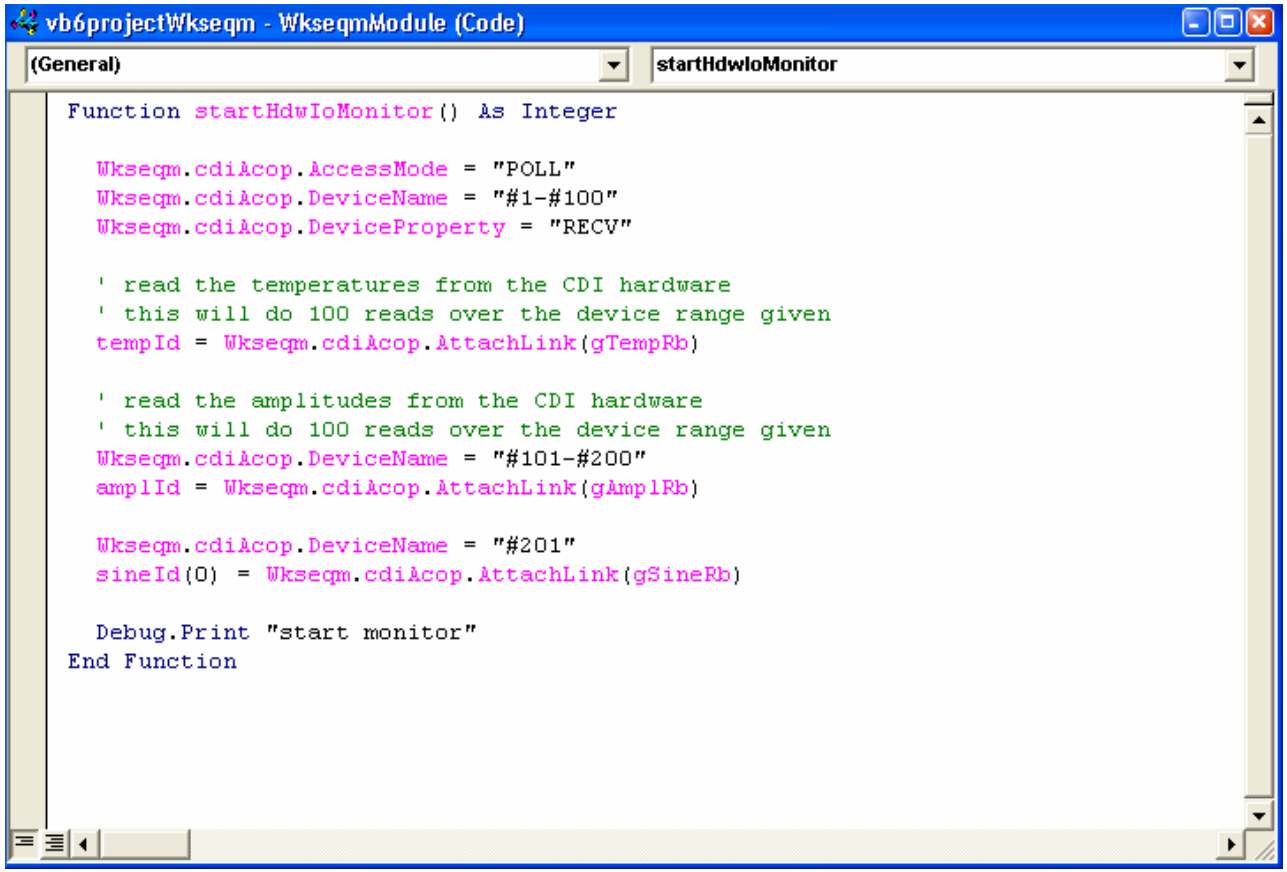
CDI Practical Example (C)



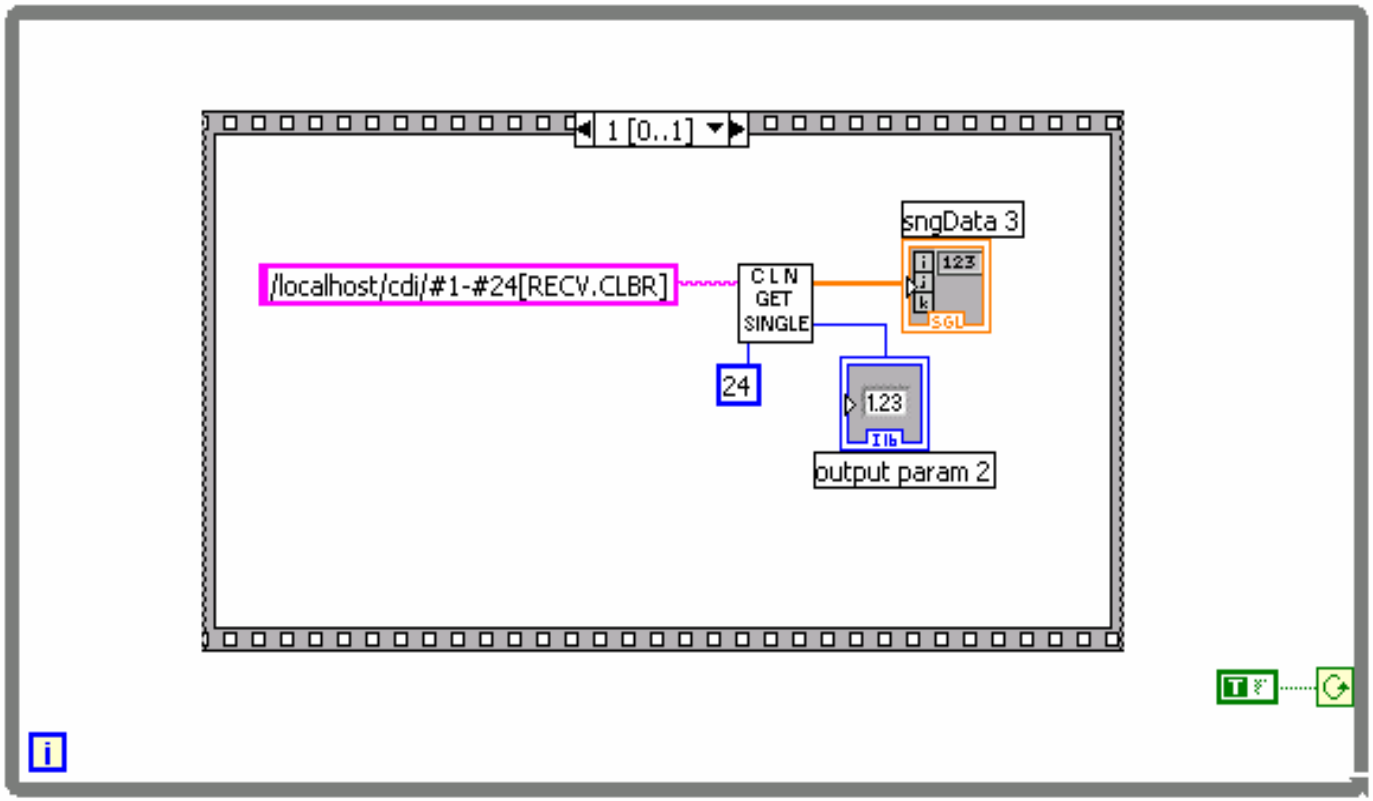
CDI: Practical Example (Java)

```
public static void start()
{ // call this test method as an alternative to a background task
  // start up CDI ...
  Cdi.start();

  // write something to some registered devices ...
  // using device numbers in the server code is probably preferable
  // as the maybe only the database knows what names were assigned ...
  setValue("#1",66);

  // start a read monitor on the temperature devices ...
  tempLink = new TLink(cdiPrefix + "#1-#100","RECV",new TDataType(tempRb),null,TAccess.CA
  HdwIoCallback tempCb = new HdwIoCallback();
  tempLink.attach(TMode.CM_POLL,tempCb,1000);
  System.out.println("temp link status : " + tempLink.getLinkStatus());

  // start a read monitor on the amplitude devices ...
  amplLink = new TLink(cdiPrefix + "#101-#200","RECV",new TDataType(amplRb),null,TAccess.
  HdwIoCallback amplCb = new HdwIoCallback();
  amplLink.attach(TMode.CM_POLL,amplCb,1000);
  System.out.println("ampl link status : " + amplLink.getLinkStatus());

  // start a read monitor on the sine devices ...
  sineLink = new TLink(cdiPrefix + "#201","RECV",new TDataType(sineRb),null,TAccess.CA_RE
  HdwIoCallback sineCb = new HdwIoCallback();
  sineLink.attach(TMode.CM_POLL,sineCb,1000);
  System.out.println("sine link status : " + sineLink.getLinkStatus());
}
```

CDI: Practical Example (VB)

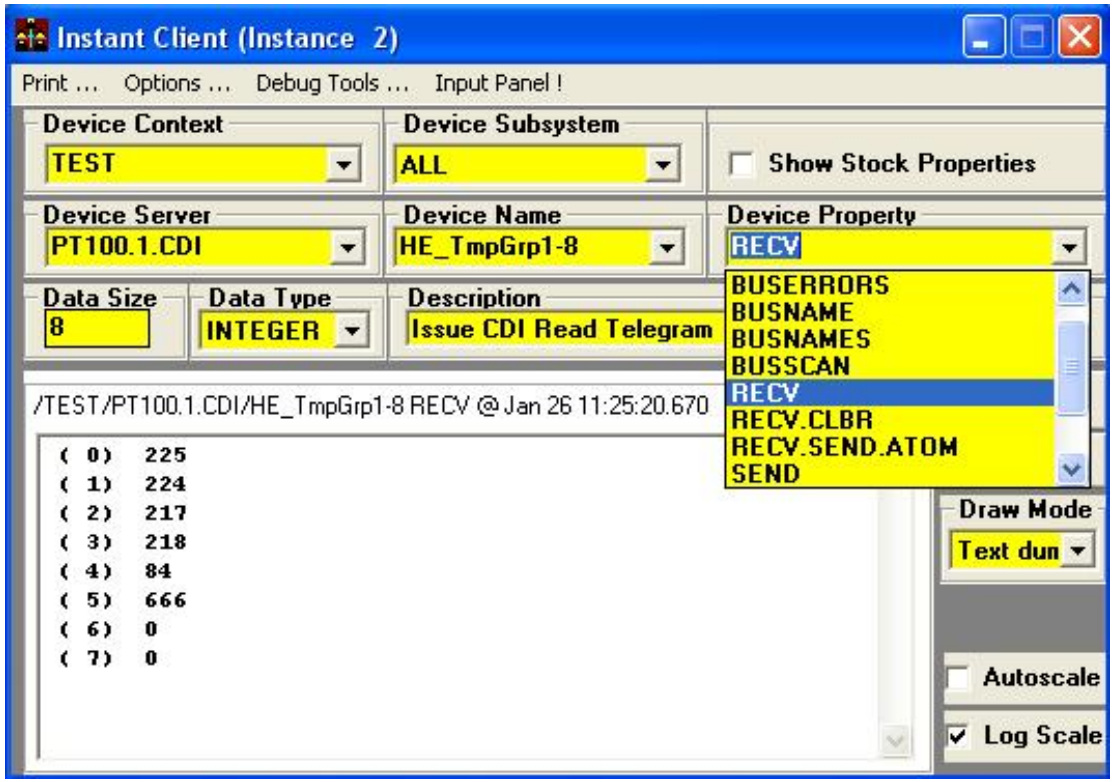


CDI: Practical Example (LabView)



CDI Remote

CDI automatically exports a (raw data) device server with the name <FECNAME>.CDI



CDI:Petra3/EMBL Motor Server with Beckhoff PLCs

- Bus Plugs (ADS DLL) for Beckhoff PLCs for Windows
- A PLC program (Motor and div IOs) in TwinCat

Use template in CDI database to register devices at initialization:

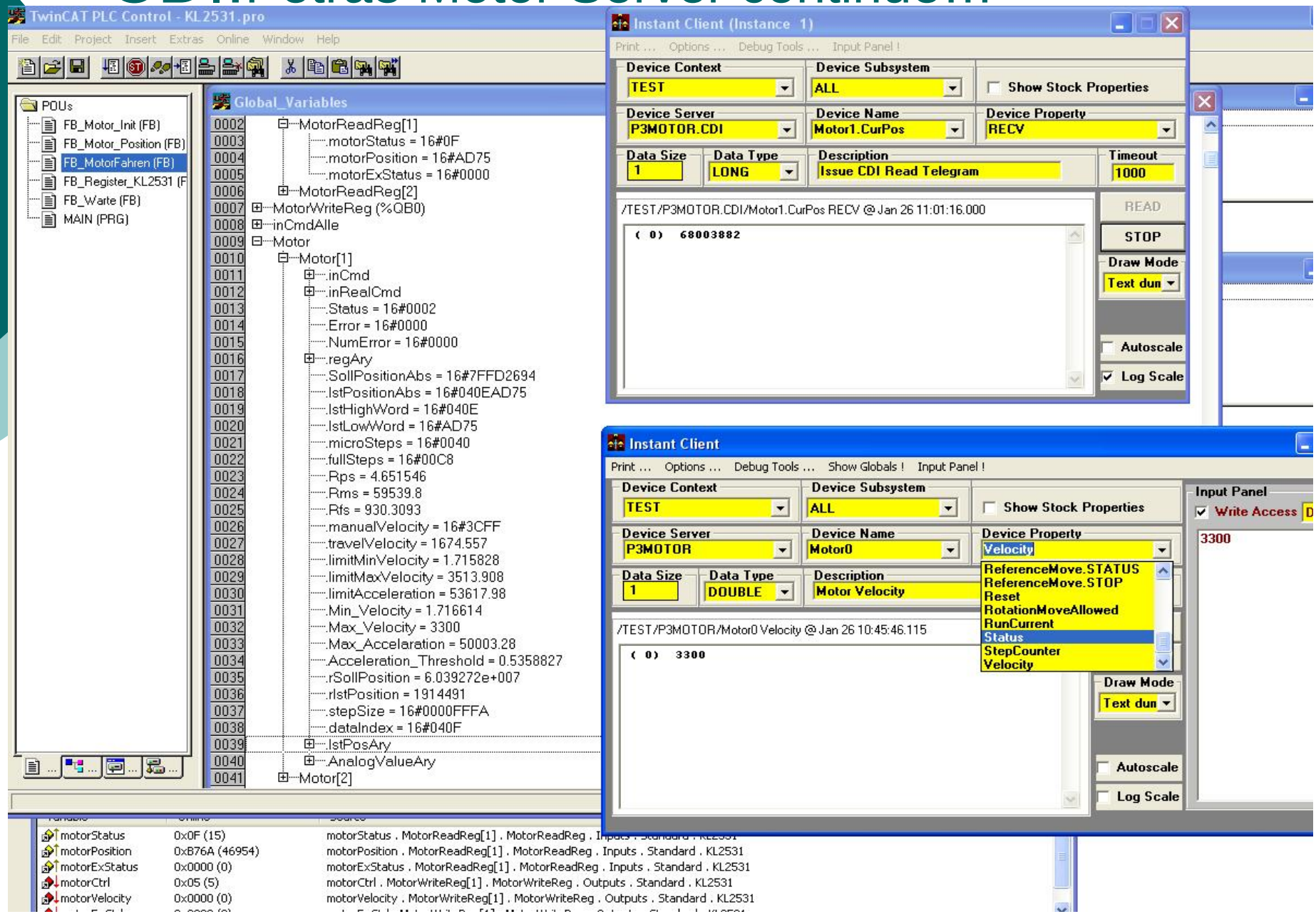
cdiaddr-llll.csv

	A	B	C	D	E	F	G	H	I	J	K	L
10	NUMBER	NAME	BUS	LINE	MASK	ADDRESS	ACCES	INPUT	FORMAT	LONG_NAME	LIMIT	PATTERN
11	0	MOTOR:Start	TEMPLATE	0					short	Motor[].Run	1:01	
12	0	MOTOR:CMD	TEMPLATE	0					long	Motor[].inCmd	6:04	!=1:04
13	0	MOTOR:fltCMD	TEMPLATE	0					float	Motor[].inRealCmd	6:04	!=1:04
14	0	MOTOR:TgtPos	TEMPLATE	0			RD		long	Motor[].SollPositionAbs	1:01	
15	0	MOTOR:CurPos	TEMPLATE	0			RD		long	Motor[].IstPositionAbs	1	
16	0	MOTOR:Type	TEMPLATE	0			RD		Short	Motor[].regAry[8]	1	
17	0	MOTOR:sync	TEMPLATE	0			RD|WR		Short	Motor[].syncRunFlag	1	
18	0	MOTOR:Status	TEMPLATE	0			RD		Short	Motor[].Status	1	
19	0	MOTOR:mSteps	TEMPLATE	0			RD		Short	Motor[].microSteps	1	
20	0	MOTOR:fSteps	TEMPLATE	0			RD		Short	Motor[].fullSteps	1	
21	0	MOTOR:Rps	TEMPLATE	0			RD		float	Motor[].Rps	1	
22	0	MOTOR:Rms	TEMPLATE	0			RD		float	Motor[].Rms	1	
23	0	MOTOR:Rfs	TEMPLATE	0			RD		float	Motor[].Rfs	1	
24	0	MOTOR:mVeloc	TEMPLATE	0			RD		float	Motor[].travelVelocity	1	
25	0	MOTOR:rCurPos	TEMPLATE	0			RD		float	Motor[].rIstPosition	1	
26	0	MOTOR:rTgtPos	TEMPLATE	0			RD		float	Motor[].rSollPosition	1	
27	0	MOTOR:maxVel	TEMPLATE	0			RD		float	Motor[].Max_Velocity	1	
28	0	MOTOR:minVel	TEMPLATE	0			RD		float	Motor[].Min_Velocity	1	
29	0	MOTOR:maxAcc	TEMPLATE	0			RD		float	Motor[].Max_Accelaration	1	
30	0	MOTOR:thrAcc	TEMPLATE	0			RD		float	Motor[].Accelaration_Threshould	1	
31	0	MOTOR:LwrLmtV	TEMPLATE	0			RD		float	Motor[].limitMinVelocity	1	
32	0	MOTOR:UprLmtV	TEMPLATE	0			RD		float	Motor[].limitMaxVelocity	1	
33	0	MOTOR:UprLmtA	TEMPLATE	0			RD		float	Motor[].limitAcceleration	1	
34	0	MOTOR:regCont	TEMPLATE	0			RD		Short	Motor[].regAry	59	
35	0	MOTOR:Error	TEMPLATE	0			RD		Short	Motor[].Error	1	
36	0	MOTOR:NumError	TEMPLATE	0					Short	Motor[].NumError	1	
37	0	MOTOR:dataIdx	TEMPLATE	0			RD		Short	Motor[].dataIndex	1	
38	0	MOTOR:regStat	TEMPLATE	0			RD		byte	MotorReadReg[].motorSTatus	1	
39	0	MOTOR:regPosi	TEMPLATE	0			RD		short	MotorReadReg[].motorPosition	1	
40	0	MOTOR:regExSta	TEMPLATE	0			RD		short	MotorReadReg[].motorExSTatus	1	
41	0	MOTOR:mData	TEMPLATE	0			RD		short	Motor[]	1024	
42	1	Motor1	TWINCAT	1		1.0:1:131:169:9:232:1:1:<MOTOR>			Short			
43	2	Motor2	TWINCAT	1		2.0:801:131:169:9:232:1:1:<MOTOR>			Short			
44	40	istAry	TWINCAT	1		1.0:1:131:169:9:232:1:1	RD		long	.IstPosAry	32768	
45	51	datAry	TWINCAT	1		1.0:1:131:169:9:232:1:1	RD		long	.analogValueAry	32768	
46	52	selMotor	TWINCAT	1		1.0:1:131:169:9:232:1:1			short	.selMotor	1	

Pattern

!=1:04

CDI: Petra3 Motor Server continue…



CDI: Current Status

- Bus Plugs for
  - SEDAC (SedPC, SedIP, SedUSB, SedISA) for Windows, Linux
  - CAN (CANOpen) for Windows, Linux
  - RS232 for Windows, Linux
  - TwinCat PLCs for Windows
  - Siemens Simatic PLCs for Windows
  - TICOM for Linux
- CDI loosely coupled to TINE
  - tine32.dll or libtine.so required
  - Can run in ‘stand-alone’ mode
- New Bus Plugs easy to create !
  - Independent of CDI Lib

System.exit(0)

CDI is plug-and-play.

Writing Bus Plugs is straightforward.

See Examples … …