Using the Common Device Interface in TINE

Philip Duval and Honggong Wu, DESY MST, Hamburg, Germany

this.Talk()

- A few words about TINE and API
- A few words about driver support
- Motivation behind CDI; Design goals
- How CDI works
- CDI manifest; CDI database
- Where we stand

A brief history of TINE

(Three-fold Integrated Networking Environment)

- CERN Isolde (1989)
- Equipment Modules (early object-oriented)
- Continuous development @ DESY since 1992
- Unix/VMS ports ~ 1993/4
- VxWorks ~1995
- Produce-Consume; Publish-Subscribe ~1995/6
- User-defined ‘tagged’ structures ~1995/6
- Publish/Consume ~ 1999
- Java ~ 1998/9
- ...

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, etc ...)
  - Epics2Tine (leverage the EPICS drivers)
  - TINE via LabView (leverage the LV drivers)
  - TINE via doocs (leverage the doocs drivers)

hardware support (2)

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

http://www.beckhoff.com

Application Programmers

- either learn an interface API for each hardware bus, or
- use Epics + Epics2Tine or
- use LabView + TINE VIs or
- use DOOCS + run DOOCS via TINE or
- use TINE CDI !

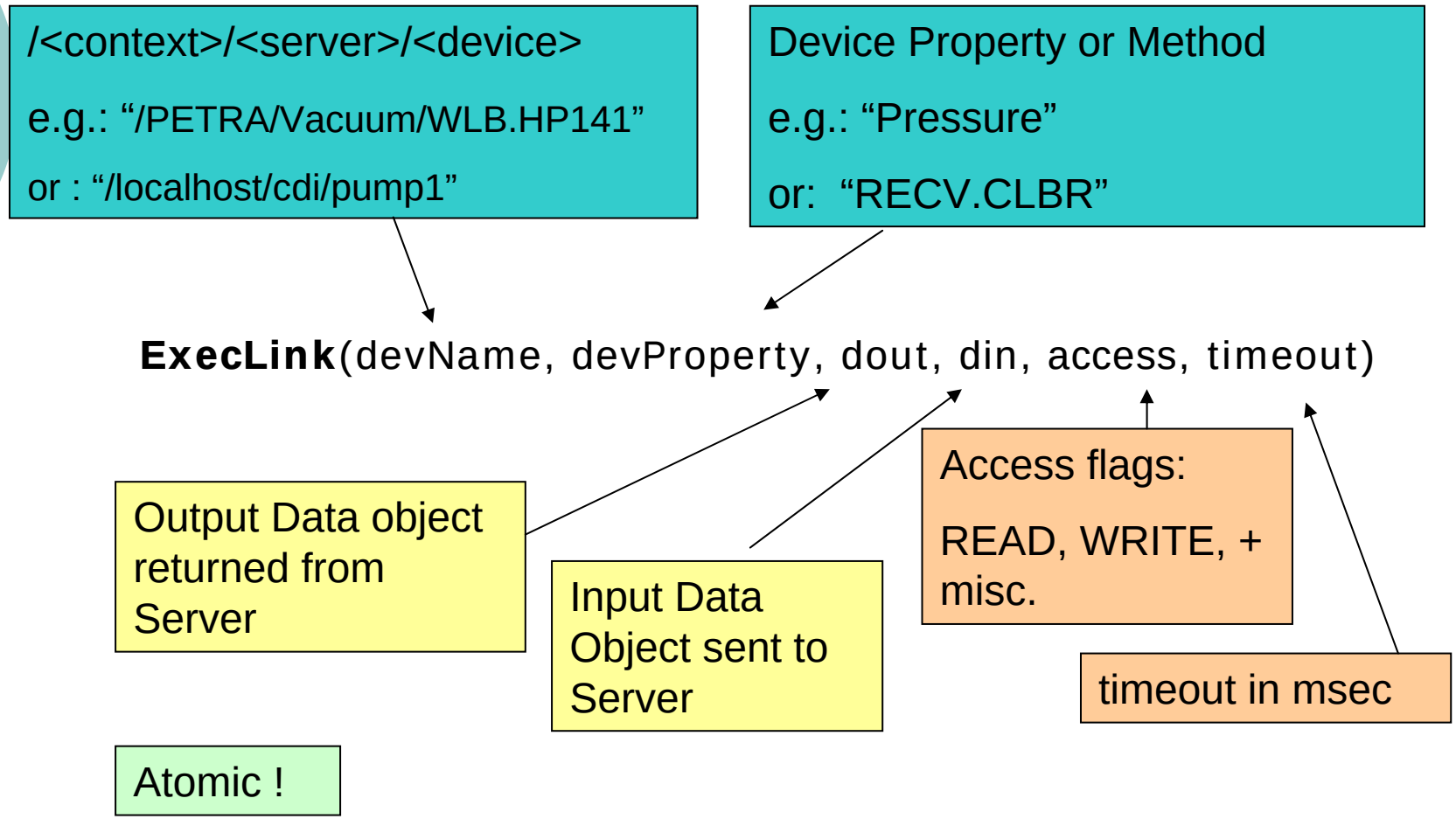
CDI API

- 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
  - native CDI API interface (TINE similar for device registration and device access)
  - direct TINE client API for accessing hardware devices + database registration of devices.

So what is the TINE client interface?

- **NOT** get(), set(), monitor() !!!!
- Endpoint:
  - /<context>/<server>/<device> [<property>]
- synchronous and asynchronous calls
  - 'calls' more in the sense of RPC, RMI
  - deal with links (some terminology left over from DDE).
- Data “Objects” can be sent to and/or received from a target.
- Atomic.

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/bpmPos1 – bpmPos100”
  - “/localhost/cdi/reset1”
  - ...
- **Device Properties** (methods)
  - “RECV”
  - “SEND”
  - “RECV.SEND.ATOM”
  - “SEND.RECV.ATOM”
  - “RECV.CLBR”
  - “SEND.RECV.CLBR”
  - “ADDR”
  - “BUSNAME”
  - ...

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 *));

…

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_FLOAT;
dout.data.fptr = rbPressData;
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.

```
dout.dArrayLength = 100;
dout.dFormat = CF_FLOAT;
dout.data.fptr = 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

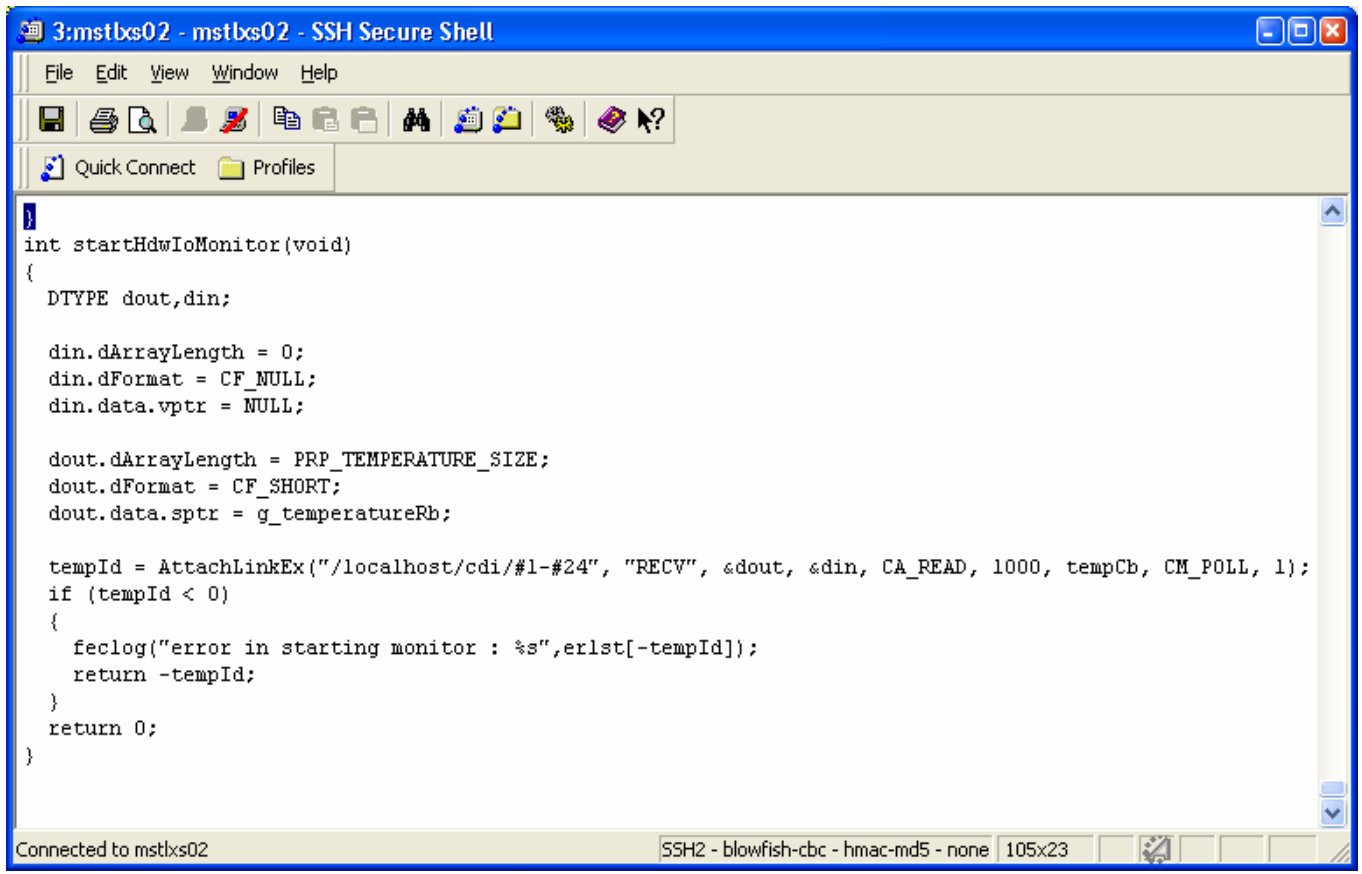
```
din.dArrayLength = 1;
din.dFormat = CF_UINT16;
din.data.sptr = &setValue;
ExecLink (“/localhost/cdi/#16”, ”SEND”, NULL, &din,1000)
```

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.

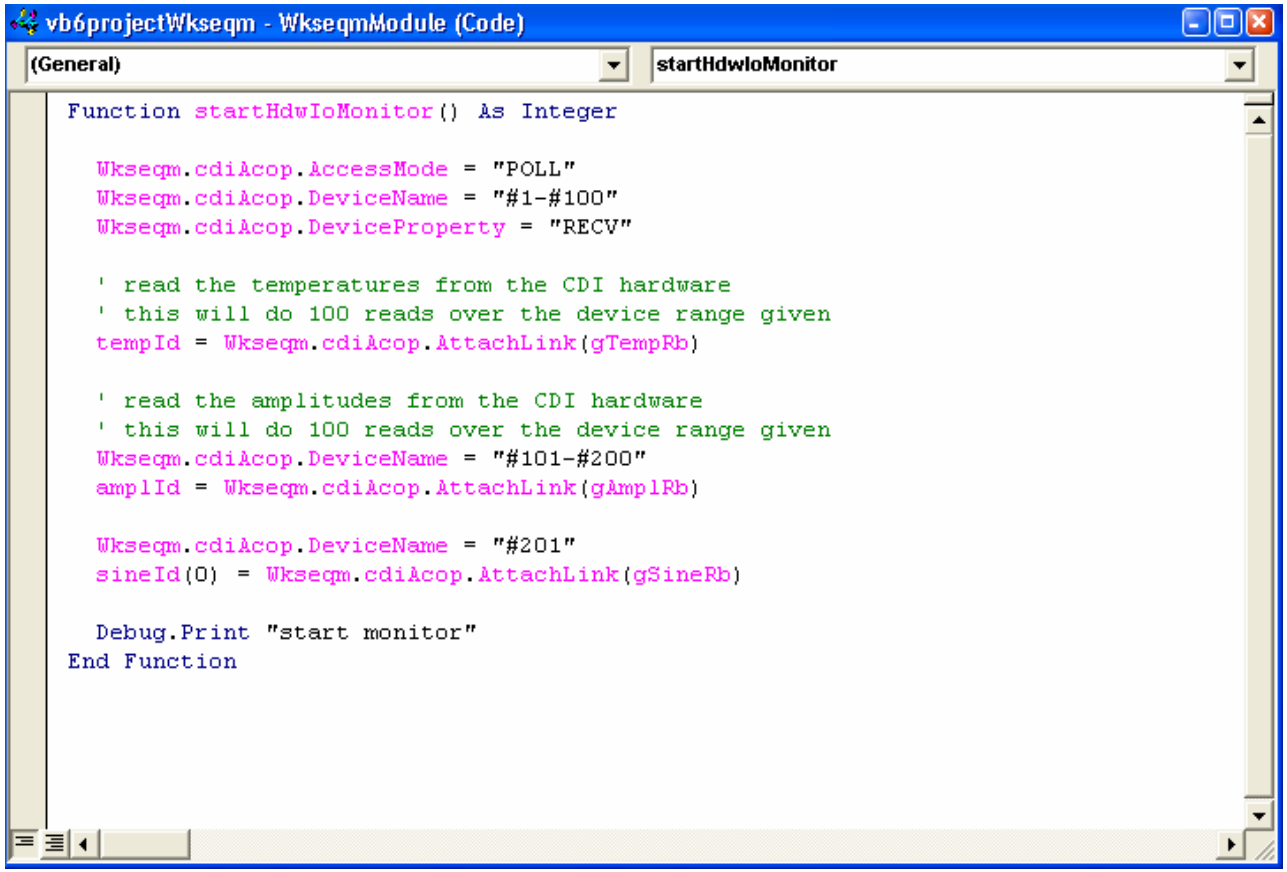
CDI Practical Example (C)



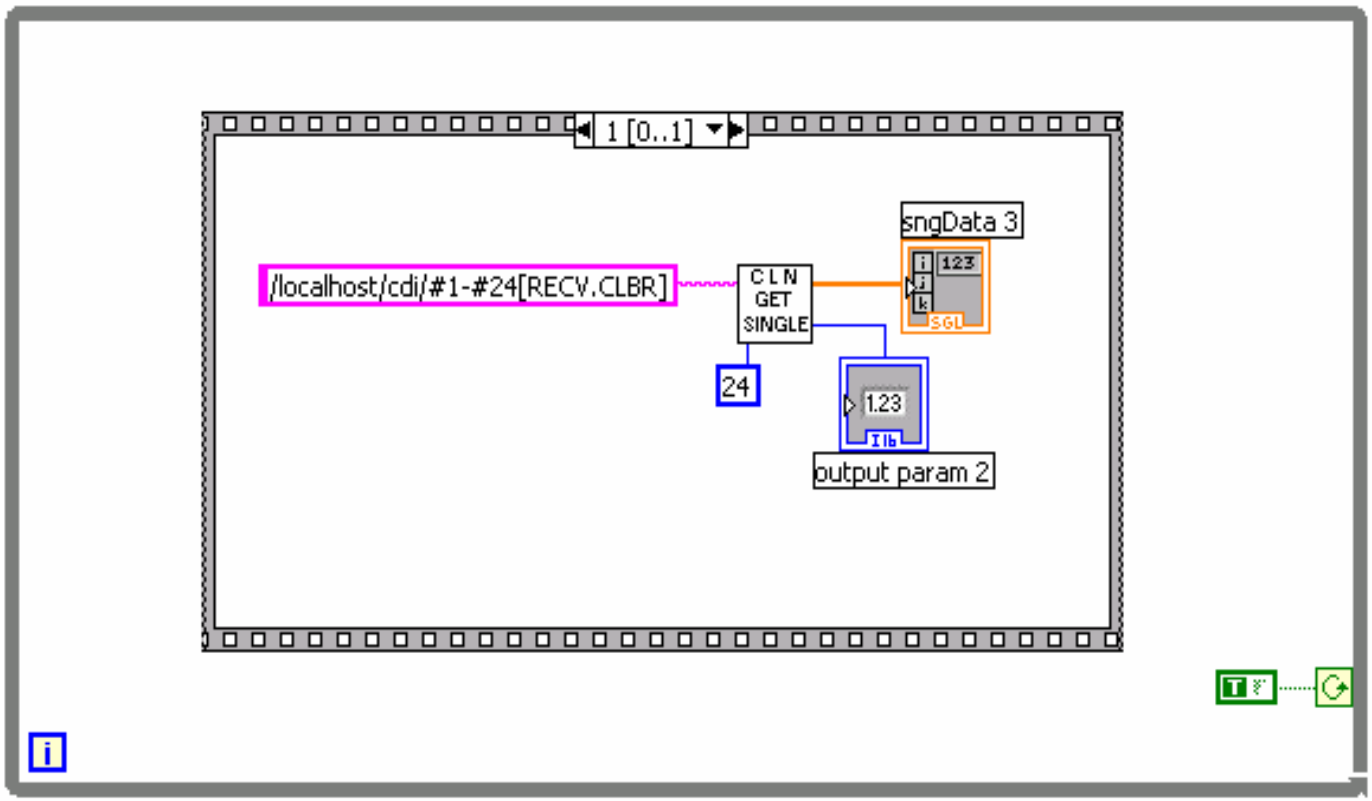
CDI: Practical Example (Java)

```
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)

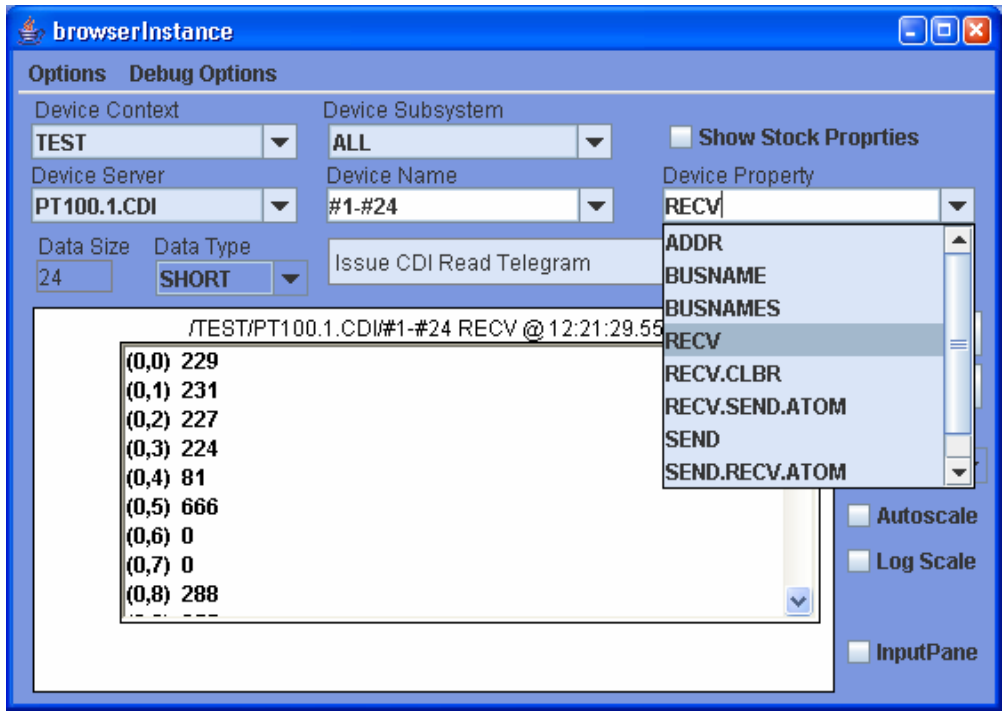


CDI: Practical Example (LabView)



CDI Remote

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



CDI: Current Status

- Bus Plugs for
  - SEDAC (SedPC, SedIP, SedISA) for Windows, Linux
  - CAN (CANOpen) for Windows, Linux
  - RS232 for Windows, Linux
  - TwinCat for Windows
- 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.

Need a database configuration tool (other than Excel)!

Need to integrate with TINE server wizard!

Ref: http://tine.desy.de

The four basic personality types