The TINE Common Device Interface in Operation

P. Duval, H. Wu, U. Ristau (+I. Kriznar, J. Bobnar)

CDI: A brief review …

- CDI library
  - Synchronous, asynchronous i/o to hardware devices
  - Multi-threaded: Each line handled in independent thread
  - Device groups can span multiple lines
- CDI bus plugs
  - Handle bus specific i/o
- CDI manifest
  - Defines managed hardware busses
- CDI database
  - Assigns device names to addresses
- CDI hook in TINE libraries
  - context = “localhost”, server = “cdi” passes directly to CDI

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’ dispatch routines into CDI:

int **cdiRegisterBus**(char *busName);
int **cdiRegisterBusInitialization**(char *busName,int (*fcn)(int,int,int,char *));
int **cdiRegisterBusHandler**(char *busName,void (*fcn)(CdiRequestInfoBlk *));

…

CDI: Sample Database …

	A	B	C	D	E	F	G	H
1	NAME	BUS	LINE	ADDRESS_BASE	ADDRESS_PARAMETERS	ADDRESS_MAP	ACCESS	FORMAT
2	MonAdc:adcSta	TEMPLATE	0	0	00:00			LONG
3	MonAdc:trgMod	TEMPLATE	0	0	00:01		WR	LONG
4	MonAdc:rstOvl	TEMPLATE	0	0	00:02		WR	LONG
5	MonAdc:rstTrg	TEMPLATE	0	0	00:03		WR	LONG
6	M2AdcPia	SEDAC	1	10	<MonAdc>			LONG
7	M4AdcPia	SEDAC	1	10.32	<MonAdc>			LONG
8	M10AdcPia	SEDAC	1	10.96	<MonAdc>			LONG
9	M12AdcPia	SEDAC	1	10.128	<MonAdc>			LONG
10	SDO1008	CANESD:125	1	5	0x0:0x1008:0x0:0:0			BYTE
11	SDO100A	CANESD:125	1	5	0x0:0x100A:0x0:0:0			BYTE
12	MOTOR:Start	TEMPLATE	0	0		Motor[].Run		short
13	MOTOR:CMD	TEMPLATE	0	0		Motor[].inCmd		long
14	MOTOR:fltCMD	TEMPLATE	0	0		Motor[].inRealCmd		float
15	MOTOR:TgtPos	TEMPLATE	0	0		Motor[].SollPositionAbs	RD	long
16	MOTOR:CurPos	TEMPLATE	0	0		Motor[].IstPositionAbs	RD	long
17	MOTOR:Type	TEMPLATE	0	0		Motor[].regAry[8]	RD	Short
18	MOTOR:sync	TEMPLATE	0	0		Motor[].syncRunFlag	RD|WR	Short
19	MOTOR:Status	TEMPLATE	0	0		Motor[].Status	RD	Short
20	MOTOR:mSteps	TEMPLATE	0	0		Motor[].microSteps	RD	Short
21	Motor1	TWINCAT	1	1.0.1.131:169:9:23	<MOTOR>			Short
22	Motor2	TWINCAT	1	2.0.801.131:169:9:	<MOTOR>			Short
23								

Optional: “RULE_RECV”, “RULE_SEND”, “MASK”, “PATTERN”, “LIMIT”, “INPUT”

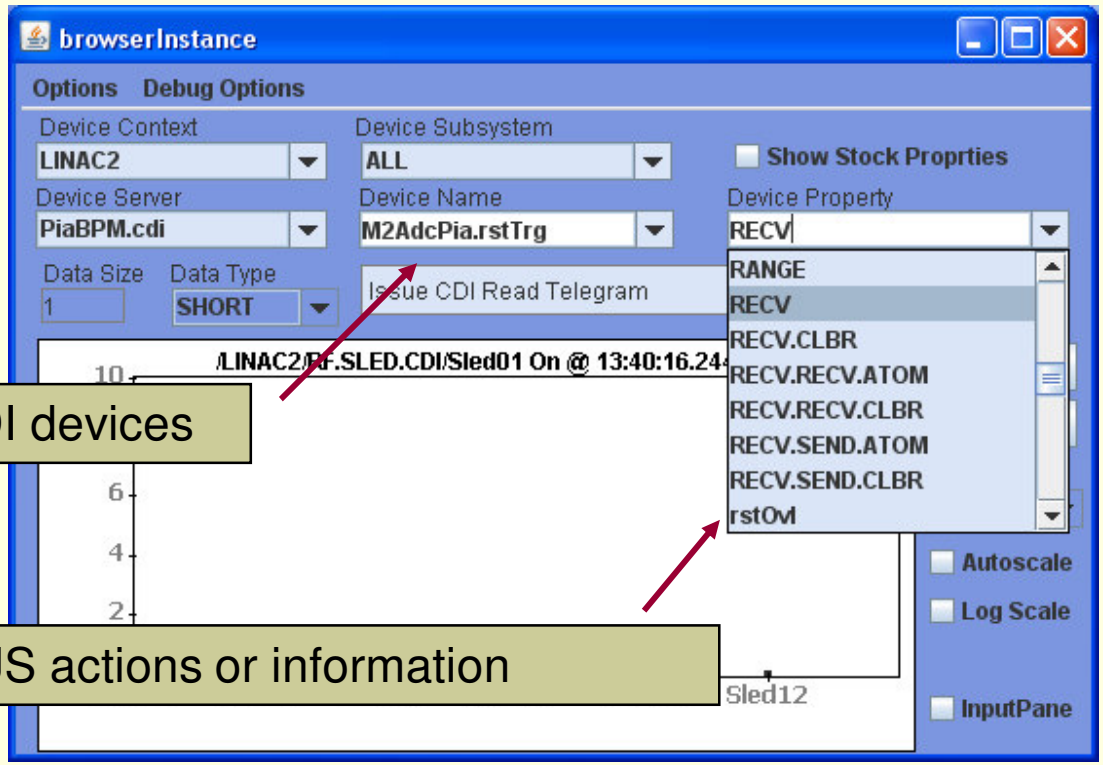
CDI Hardware Server …

TINE Automatically supplies a CDI “hardware” server !

Local (in-process) access uses :

Context = “localhost”

Server =“cdi”



Devices = CDI devices

Properties = BUS actions or information

CDI New Features

- TINE Release 4 compliant
  - Longer names!
    - CDI device names
      - up to 32 characters
      - but can specify and address a device “group”
        - **“M2AdcPia.rstTrg, M10AdcPia.rstTrg, M18AddPia.dbSta - M26AddPia.dbSta”**
    - TINE:
      - registered device names 64 characters
      - device name field can carry 1024 characters
  - Templates
    - define and register instances of address patterns
  - Bitfields
    - name any bit or bit pattern
    - registered like templates (use “MASK” information)
    - templates can contain bitfields
  - Group or Single Reads
    - can ask for (array of) simple format (e.g. int or float)
    - can ask for (array of) value-status pairs
    - can ask for (array of) name-value-status triplets

CDI Templates and Bitfields

Bitfield "READ"

Template "TrgtRvlvr"

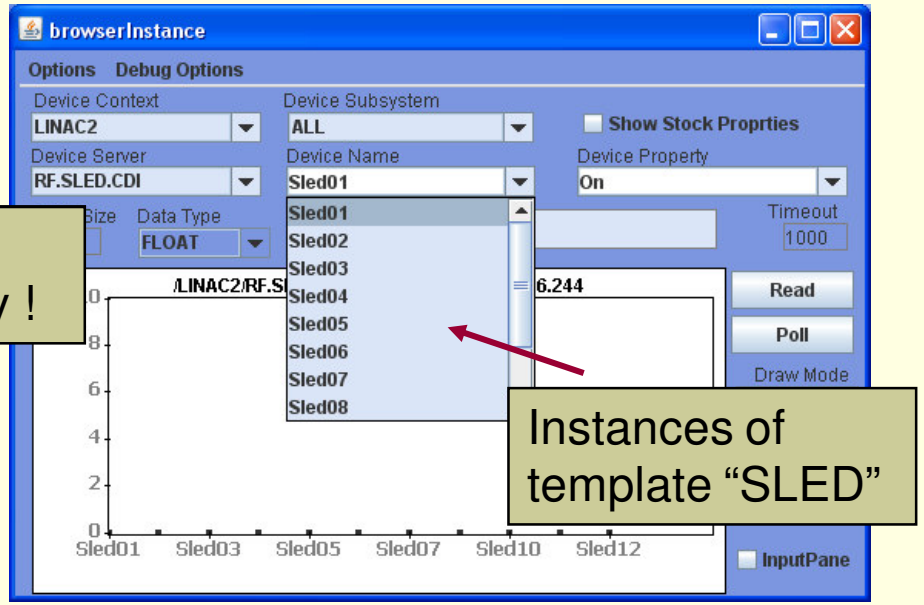
1	NAME	BUS	LINE	ADDRESS_BASE	ADDRESS_PARAM	FORMAT	ACCESS	INPUT	MASK
2	SEDPC:Teststrahltargets	FIELDBUS	1			Short			
3	READ:T1IstInPos	BITFIELD	0			Short			0x001
4	READ:T2IstInPos	BITFIELD	0			Short			0x002
5	READ:T3IstInPos	BITFIELD	0			Short			0x004
6	READ:T4IstInPos	BITFIELD	0			Short			0x008
7	READ:T5IstInPos	BITFIELD	0			Short			0x010
8	READ:T6IstInPos	BITFIELD	0			Short			0x020
9	READ:PowerOK	BITFIELD	0			Short			0x040
10	READ:TrgtWchsIt	BITFIELD	0			Short			0x080
11	TrgtRvlvr:T1InPos	TEMPLATE	0		00:01	Short	WR	1	
12	TrgtRvlvr:T2InPos	TEMPLATE	0		00:01	Short	WR	2	
13	TrgtRvlvr:T3InPos	TEMPLATE	0		00:01	Short	WR	4	
14	TrgtRvlvr:T4InPos	TEMPLATE	0		00:01	Short	WR	8	
15	TrgtRvlvr:T5InPos	TEMPLATE	0		00:01	Short	WR	16	
16	TrgtRvlvr:T6InPos	TEMPLATE	0		00:01	Short	WR	32	
17	TrgtRvlvr:Status	TEMPLATE	0		00:02	BITFIELD16:<READ>			
18	SpllDspl:S21	TEMPLATE	0		00:01	Short	WR	1	
19	SpllDspl:S22	TEMPLATE	0		00:02	Short	WR	2	
20	SpllDspl:S24	TEMPLATE	0		00:03	Short	WR	4	
21	S21Trgt	SEDPC	1	6.112	<TrgtRvlvr>	Short			
22	S22Trgt	SEDPC	1	6.128	<TrgtRvlvr>	Short			
23	S24Trgt	SEDPC	1	6.144	<TrgtRvlvr>	Short			
24	Spill	SEDPC	1	6.96	<SpllDspl>	Short			

e.g. Device name : S21Trgt.Status.PowerOK

CDI New Features …

- Templates
  - Imagine …
    - PSC module with 30 address registers
    - 300 PSCs -> 9000 database entries ?
    - PSC template + 300 PSCs -> 330 database entries!
  - CDI device names
    - <instance>.<field>

<template field> ~ attribute
=> also registered as property !



Instances of template “SLED”

CDI New Features

- Calibration rules
  - RULE_RECV
    - Mathematical operations (the usual stuff)
    - Bit operations (XOR, AND, NOT, shift, etc.)
    - Message (convert to text according to bit pattern)
    - External function
    - Applies to “RECV.CLBR” (and template field properties)
  - SEND_RECV
    - Mathematical operations
    - Bit operations
    - de-calibrate prior to sending out
    - Applies to “SEND.CLBR” (and template field properties)

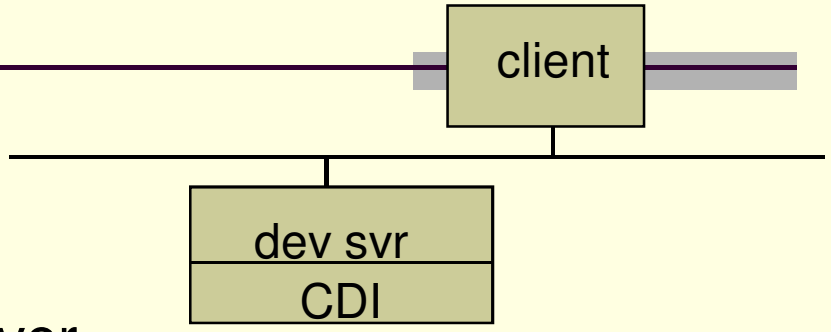
CDI New Features

- BUS Names
  - What bus is “M2AdcPia.rstTrg” on ?
  - CDI knows automatically something like “SEDUSB-line1”
    - i.e. the SEDUSB line 1 attached to the FEC.
  - But this can be given a more descriptive name!
    - e.g. “PiaBPMs” (a more appropriate field bus name)

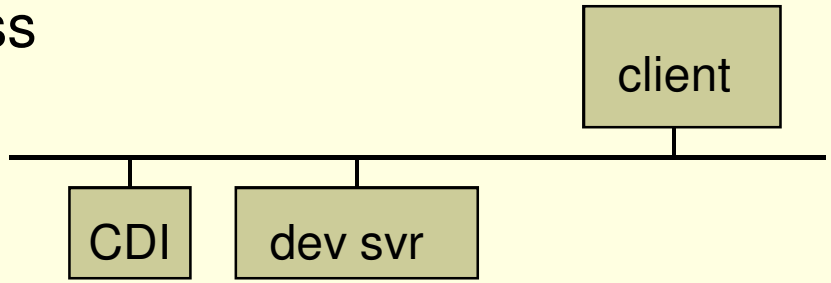
CDI Database Manager

- Currently
  - Make the database spreadsheet by hand !
  - Copy-and-paste-and-edit is error prone
    - (cdi.log will tell you what's wrong)
  - But: there is a database consistency checker!
  - COSYLAB (Igor) is 'almost finished' with a real CDI database manager !

CDI and Device Servers …



- Best practice …
  - create CDI database
  - test the hardware i/o
  - write/generate device server
    - CDI device names may or may not have anything to do with server device names
      - D2BPM.cdi/M1AdcD2.adcSta -> DESY2/BPM/NW24
  - use CDI in-process to access the hardware
- But often …
  - use CDI hardware server as device server
    - ~ 50 % !
  - use CDI out-of-process
    - the other ~ 50%

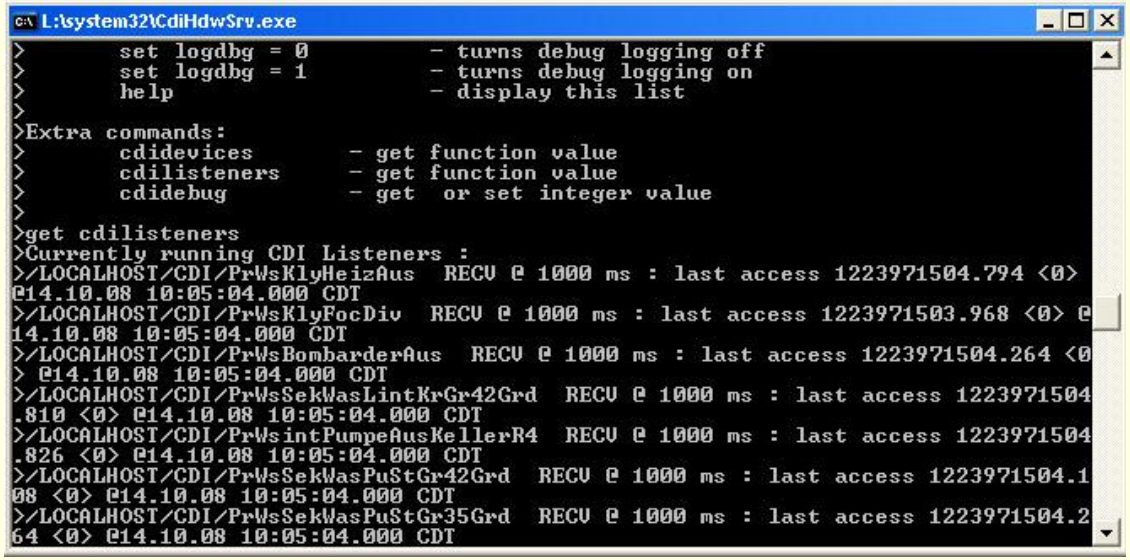


CDI + TINE

- 1) Device Server activates the CDI hook …
  - TINE client API
  - Calls to "/localhost/cdi/…" feed thru to CDI
  - Calls to "/<*context*>/<*cdi server*>/…" also feed thru to CDI if
    - <*context*> = <*my context*>
    - <*cdi server*> = <*my cdi server*>
  - in-process
- or 2) Device Server does not activate CDI hook …
  - Calls to "/localhost/cdi/…" fail
  - Calls to "/<*context*>/<*cdi server*>/…" find the server and behave normally
  - out-of-process (i.e. remote)

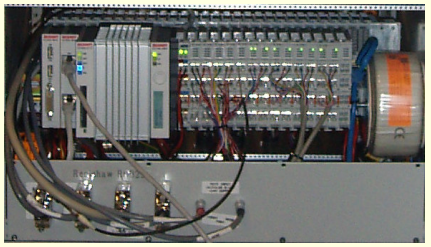
Using the CDI Hardware server

- Restrict write access !
- Asynchronous data acquisition !
  - asynchronous data links start asynchronous cdi “listeners” !
    - all clients for a data readback collapse to a single listener.
    - profit most efficiently the from CDI threading model.
- Synchronous single transactions



CDI in Action …

- LINAC2, DESY2, PIA using CDI for operations since July/August
  - 35 CDI servers
    - Mostly varieties of CAN or SEDAC
    - Equal mix of Win XP and Linux or ELINOS
  - ~ 15 with direct links to console applications
- EMBL
  - Multi-layer Monochromator
  - Motor control
  - TwinCat (Beckoff)
- Now running smoothly !
  - Concurrency problems eliminated
  - Most recent bug: millisecond counter wraps every 24 days!



Summary

- All desired functionality in place !
- Used in operations since July 2008
- Many CDI “hardware” servers in use as de-facto device servers
- CDI “loosely” coupled to TINE
- TO-DO List
  - Finish database manager
  - Documentation !

- http://tine.desy.de -> CDI