# THE TINE COMMON DEVICE INTERFACE IN OPERATION

Philip Duval, Honggong Wu, DESY, Hamburg
Uwe Ristau, EMBL Hamburg

*Abstract*

The Common Device Interface (CDI) [1] is the primary device layer used in the TINE [2] control system. It offers a generic, database-driven view of a server's hardware, where a hardware address, irrespective of the underlying bus, simply appears as a named device, which is accessed via the TINE client API. To date, CDI-supported busses include several CAN implementations, RS232, TwinCat [3], Libera [4], Siemens PLC, as well as the DESY in-house bus SEDAC. In this paper, we report on the latest features of CDI and more importantly on the first experiences of using CDI in operations, primarily in the PETRA3 pre-accelerator chain and in DC, Servo, and stepper motor control at the EMBL beam lines.

## INTRODUCTION

The Common Device Interface was first presented in significant detail at PCaPAC 2006 at which time it was undergoing test commissioning on a few selected subsystems. Since that time it has undergone extensive tests and is now in use in the field in the pre-accelerator chain for DORIS and PETRA3, namely DESY2, LINAC2, and PIA as well as in motor control at the EMBL Hamburg beam lines. Numerous new features, bug fixes, and general experience have ensued during the course of the past two years. These will all be highlighted below. But first we digress into a brief review.

CDI is loosely coupled to TINE [2] in that although it makes use of routines from the TINE library, it does not itself depend on the TINE protocol. Nonetheless it offers a CDI-native API which is TINE-similar, and by and large TINE developers will want to make use of precisely the same TINE client API calls as used when accessing data from any other end-point in the control system.

CDI operates on a plug-and-play basis, where adding a new bus interface plug to CDI only involves writing a new bus interface plug. The CDI shared library needs only to be compiled and installed once for the platform in question. When the library loads, it will look for a CDI bus manifest file which provides a list of bus plug libraries, which are also loaded. If the bus-plug library loads successfully, it will register its name and all dispatch handlers with CDI, which itself has no a priori knowledge of any hardware bus interface.

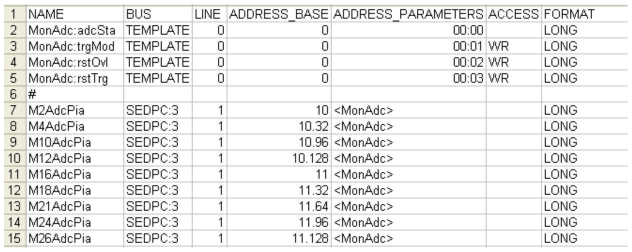
After the manifest has been read, CDI reads a CDI device database and registers all templates, devices and device information contained within. Essentially, one can name and access hardware addresses in a common manner via CDI, and by applying the appropriate masks, bitfields, and calibration rules one can sometimes go rather far in supplying a 'finished' device server by configuring a CDI database. This is especially true when CDI is used in a TINE control system, in which case a de-facto 'hardware' device server is automatically generated providing a direct TINE interface to the hardware, where the CDI devices appear as TINE devices and bus operations appear as TINE properties.

## NEW FEATURES

### *Templates*

One of the most important and sought after features added to CDI since last reported is the addition of templates to the database. Imagine a power supply controller (PSC) with 30 addressable registers each receiving a CDI device name in the CDI database. Then having 100 such PSCs would require 3000 individual entries in the database! Instead it is now possible to identify a PSC type via a template providing field names for each of the address register offsets, and a template name (e.g. "PSC") describing the template. Then the 100 PSCs can be entered as 100 individual entries, each of type "PSC". The individual CDI devices and device names are automatically generated, with names such as "psc1.status" or "psc2.setpoint", etc. depending on the template field names ("status", "setpoint", etc.) and PSC names ("psc1", "psc2", etc.).

The DC, Servo, and stepper motor control at the EMBL makes heavy use of CDI Templates, where the addition of new controllable stepper motor to the system is merely an extra line in the CDI database.



| 1 | NAME | BUS | LINE | ADDRESS_BASE | ADDRESS_PARAMETERS | 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 | # | | | | | | |
| 7 | M2AdcPia | SEDPC:3 | 1 | 10 | <MonAdc> | | LONG |
| 8 | M4AdcPia | SEDPC:3 | 1 | 10.32 | <MonAdc> | | LONG |
| 9 | M10AdcPia | SEDPC:3 | 1 | 10.96 | <MonAdc> | | LONG |
| 10 | M12AdcPia | SEDPC:3 | 1 | 10.128 | <MonAdc> | | LONG |
| 11 | M16AdcPia | SEDPC:3 | 1 | 11 | <MonAdc> | | LONG |
| 12 | M18AdcPia | SEDPC:3 | 1 | 11.32 | <MonAdc> | | LONG |
| 13 | M21AdcPia | SEDPC:3 | 1 | 11.64 | <MonAdc> | | LONG |
| 14 | M24AdcPia | SEDPC:3 | 1 | 11.96 | <MonAdc> | | LONG |
| 15 | M26AdcPia | SEDPC:3 | 1 | 11.128 | <MonAdc> | | LONG |

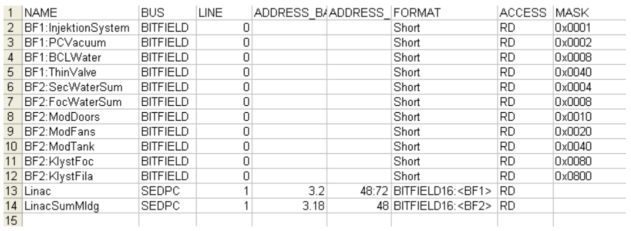
Figure 1: An example CDI database showing a template entry, MonAdc and several device modules which use this template.

### *Calibration Rules*

The original set of CDI device calibration rules has been significantly expanded, and now includes the possibility of 'reverse-'calibrating bus input data. One is, of course, always able to access the raw, unadulterated data from the hardware via CDI, if necessary. However, a 'finished' number from the hardware can often be obtained by applying a set of simple operations configured in the database. In addition to the standard arithmetic operations, these operations now include all manner of bit operations. An external function library (if included in the CDI manifest) can also provide a calibration dispatch routine which can be as complex as desired. Calibration rules however do have the constraint that the calibration operations can only manipulate a single data item read from the bus, and therefore are independent of any other data read from the bus (i.e. one cannot calibrate "a" based on the value read from "b"). In addition care must be taken to make use of bit operations and arithmetic operations in a consistent manner.

## *TINE Release 4 Compliant*

CDI is now fully compliant with TINE Release 4. Among other things, this means that the length of a CDI device name can now be 32 characters. TINE accepts up to 64 character names, but due to CDI group access, a limit of 32 characters was imposed. In addition, CDI also accepts bitfield entries in the CDI database. By using a bitfield, any bit or bit pattern read from the bus can be identified and addressed by name. To be sure, the data item read from the bus is held as an integer type and can also be read back as such if required. In analogy with templates, bitfields are named and registered inside the database, and can themselves be used within a template. Registered bitfield devices appear as CDI devices decorated with the bitfield field names. The un-decorated device will deliver the entire integer to the caller. In any case, the entire bitfield integer is always transferred, the appropriate masks and shifts occurring on the client-side. Multiple calls to the same bitfield entity collapse to a single call. The use of bitfields in write operations is not directly supported, as it is impossible for CDI to know a priori how to deal with those bits not contained in the input bitfield. Nonetheless the database can be configured where applicable to assign the same bitfield-decorated device name to input registers (with the appropriate masks) so that in many cases named bitfields can also be used in write operations.



| 1 | NAME | BUS | LINE | ADDRESS_B/ | ADDRESS_ | FORMAT | ACCESS | MASK |
|---|---|---|---|---|---|---|---|---|
| 2 | BF1:InjektionSystem | BITFIELD | 0 | | | Short | RD | 0x0001 |
| 3 | BF1:PCVacuum | BITFIELD | 0 | | | Short | RD | 0x0002 |
| 4 | BF1:BCLWater | BITFIELD | 0 | | | Short | RD | 0x0008 |
| 5 | BF1:ThinValve | BITFIELD | 0 | | | Short | RD | 0x0040 |
| 6 | BF2:SecWaterSum | BITFIELD | 0 | | | Short | RD | 0x0004 |
| 7 | BF2:FocWaterSum | BITFIELD | 0 | | | Short | RD | 0x0008 |
| 8 | BF2:ModDoors | BITFIELD | 0 | | | Short | RD | 0x0010 |
| 9 | BF2:ModFans | BITFIELD | 0 | | | Short | RD | 0x0020 |
| 10 | BF2:ModTank | BITFIELD | 0 | | | Short | RD | 0x0040 |
| 11 | BF2:KlystFoc | BITFIELD | 0 | | | Short | RD | 0x0080 |
| 12 | BF2:KlystFila | BITFIELD | 0 | | | Short | RD | 0x0800 |
| 13 | Linac | SEDPC | 1 | 3.2 | 48:72 | BITFIELD16:<BF1> | RD | |
| 14 | LinacSumMldg | SEDPC | 1 | 3.18 | 48 | BITFIELD16:<BF2> | RD | |
| 15 | | | | | | | | |

Figure 2: An example CDI database showing two bitfield entries, BF1 and BF2 as well as two device modules which assign their bus readback values to these bitfields.

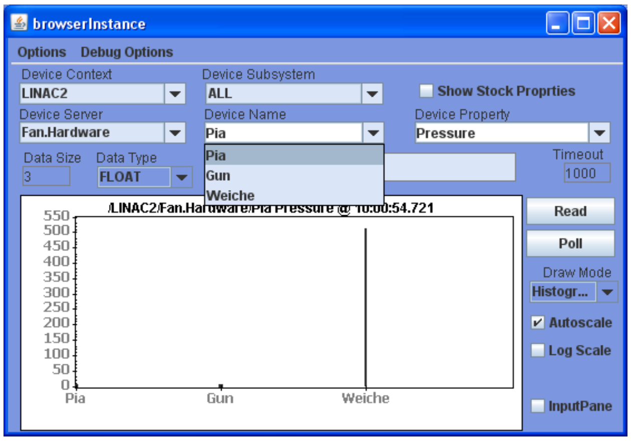
## *Bus Names*

A trivial but import new feature to CDI is the ability to name a field bus line. Consequently the bus 'name' that a particular device lives on can be queried. If no information is supplied in the CDI database to this end, the answer might be something on the order of 'CANPeak.Line1', which would essentially be the same answer given for a query of the bus 'type', which at least identifies the device as residing on the CAN bus line 1. However, the CDI database can now supply more informative names such as 'PSCs-North-hall', which might assist in trouble shooting.

## *Property-Query Precedence*

As noted above, when used with the TINE control system, CDI offers an auto-generated device server, whereby the CDI devices can be accessed via a set of standard bus operation properties (i.e. methods). Primarily, the standard set of CDI 'hardware' server properties include the bus operations "SEND" and "RECV" (as well as various atomic operations involving both "SEND" and "RECV" in combination) along with properties delivering pure information (e.g. "BUSADDR", "BUSNAME", etc.). At first glance, this would appear to yield a classic device server where the 'devices' are simply given by the complete set of CDI devices and each of these supports all available properties. However, consider the following: CDI template device names are composed of a base name decorated with the template fields which individually respond to bus properties such as "RECV" or "SEND". In most cases, the template field can be regarded as an attribute of the base device. Therefore, CDI also registers the template fields as properties, which are valid only for those base devices which make use of the template.

In TINE this amounts to using 'property-query precedence', which means that the CDI 'hardware' server in principle offers a different 'device' list for each selected property. Thus if one or more templates have a field 'status', then a property 'status' is registered, and if selected in a browser will instruct the browser to acquire all (base) devices which support this property.



browserInstance

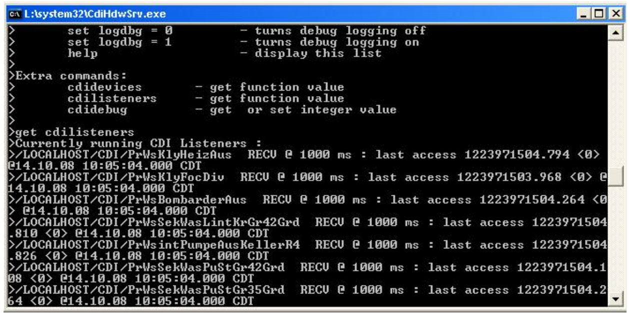
Device Context: LINAC2
Device Subsystem: ALL
Show Stock Proprties
Device Server: Fan.Hardware
Device Name: Pia
Device Property: Pressure
Data Size: 3
Data Type: FLOAT
Timeout: 1000
/LINAC2/Fan.Hardware/Pia Pressure @ 10:00:54.721
Draw Mode: Histogr...
[✓ Autoscale]
Log Scale
InputPane

Figure 3: Browsing a CDI hardware server with the TINE 'Instant Client' selects a template field Pressure which shows up as a property applicable to three 'devices' : Pia, Gun, and Weiche.

In such cases, a call to a template instance 'device' plus template field 'property' is fundamentally equivalent to calling device <instance>.<field> plus property "RECV.CLBR" (read operation) or property "SEND.CLBR" (write operation).

### *New Diagnostics*

The CDI hardware server now offers several properties which aid in diagnosing and tracking hardware problems. These include such properties as "BUSSCAN" (scan the given bus for attached hardware), "BUSERRORS" (display current bus error statistics), and "LINESTATUS" (give current hardware line information). Similarly the hardware server itself directly offers diagnostic functions at the command line.



```
L:\system32\CdiHdwSrv.exe
>       set logdbg = 0          - turns debug logging off
>       set logdbg = 1          - turns debug logging on
>       help                    - display this list
>
>Extra commands:
>       cdidevices        - get function value
>       cdilisteners      - get function value
>       cdidebug          - get  or set integer value
>
>get cdilisteners
>Currently running CDI Listeners :
>/LOCALHOST/CDI/PrWsKlyHeizAus  RECV @ 1000 ms : last access 1223971504.794 <0>
@14.10.08 10:05:04.000 CDT
>/LOCALHOST/CDI/PrWsKlyFocDiv  RECV @ 1000 ms : last access 1223971503.968 <0> @
14.10.08 10:05:04.000 CDT
>/LOCALHOST/CDI/PrWsBombarderAus  RECV @ 1000 ms : last access 1223971504.264 <0
> @14.10.08 10:05:04.000 CDT
>/LOCALHOST/CDI/PrWsSekWasLintKrGr42Grd  RECV @ 1000 ms : last access 1223971504
.810 <0> @14.10.08 10:05:04.000 CDT
>/LOCALHOST/CDI/PrWsintPumpeAusKellerR4  RECV @ 1000 ms : last access 1223971504
.826 <0> @14.10.08 10:05:04.000 CDT
>/LOCALHOST/CDI/PrWsSekWasPuStGr42Grd  RECV @ 1000 ms : last access 1223971504.1
08 <0> @14.10.08 10:05:04.000 CDT
>/LOCALHOST/CDI/PrWsSekWasPuStGr35Grd  RECV @ 1000 ms : last access 1223971504.2
64 <0> @14.10.08 10:05:04.000 CDT
```

Figure 4: Getting diagnostic information at the command line from a running CDI server. The above is running in the foreground on windows. However even if running in the background on Linux or windows, the CDI server can be attached (using the TINE attachfec utility) to yield the same diagnostic information.

### *Database Managers*

Under construction at the moment is a database manager to help create and maintain the CDI database. Currently the server developer must use a spreadsheet application (such as EXCEL) to manage a CDI database. As the possibility for introducing inadvertent errors (through copy and paste) is non-negligible, such a database manager will be a welcome utility. However, a database consistency checker does exist.

## CDI IN ACTION

The PETRA3 accelerator is scheduled for commissioning beginning 2009. Currently the pre-accelerators LINAC2, DESY2 and PIA are making extensive use of CDI in the field and serving as a test bed for tracking down problems, analyzing use-cases, and identifying missing features. With these three accelerators, there are approximately 35 CDI servers in operation, providing a hardware interface for most aspects of the control systems, from magnets to RF modulators. These servers are PCs running primarily on Windows XP or Linux, or PC104 cards running embedded Linux (ELINOS). In most cases the servers deal with CanOpen devices or various flavors of the in-house serial bus SEDAC, and in some instances, a bus plug to a Siemens PLC. In PETRA3 accelerator control, it is planned to use the CDI TwinCat bus plug for motor control. Indeed this is already the case at EMBL-Hamburg. In the latter case, both TINE and CDI are running embedded on WindowsCE.

The original conceptual design of CDI assumes that CDI constitutes a common device access layer for a TINE device server, where the necessary business logic, synchronization, process control, etc. takes place. In practice we have seen that, due to templates and the powerful calibration features of CDI, roughly half of the CDI servers being used in LINAC2 and DESY2 have direct interfaces to client-side control applications. This is particularly true for the RF subsystems.

During the initial commissioning phase of LINAC2 in June, some concurrency problems involving asynchronous grouped data acquisition were identified and resolved. During this same period, CanOpen driver problems on the PC104 were also identified, and as TINE Release 4.0 was being commissioned essentially simultaneously, assigning 'blame' for system failures was sometimes a challenging and trying experience!

Operations in these accelerators have been remarkably smooth since August of this year. In fact, the most recent bug discovered in the CDI package involved a millisecond counter that wrapped every 24 days following server start. This bug was of course immediately isolated and fixed, but required itself a certain level of stability in order to surface!

## CONCLUSION

CDI is still a 'work-in-progress', but has achieved the required level of stability for operations. The immediate 'TO-DO' list includes commissioning the CDI database manager and providing documentation for CDI on the TINE web page. Both of these aspects should be finished (to first order) within the next three months.

## REFERENCES

[1] Duval and Wu, "Using the Common Device Interface in TINE", Proceedings PCaPAC 2006.
[2] http://tine.desy.de
[3] http://www.beckhoffautomation.com.
[4] http://www.i-tech.si/products.php