



CDI News

# [ New Features ... ]

- CDI Property “CDIVERSION”
  - “1.0.0”
- Heap Damage (if detected) now in cdi.log
  - check after all dispatch routines
  - Windows only
- And ...

# CDI Documentation

General	APIs	Services	Examples & Help	Workshops & Tutorials	Low Level Support
<a href="#">Bird's Eye View</a>	<a href="#">C API</a>	<a href="#">Alarm System</a>	<a href="#">Getting Started</a>	<a href="#">TINE Workshop 2007</a>	<a href="#">Network Queue</a>
<b>Overview</b>	<a href="#">Visual Basic API</a>	<a href="#">Archive System</a>	<a href="#">TINE Server Wizard</a>	<a href="#">Quick Tutorial (Windows)</a>	<a href="#">Common Device Interface (CDI)</a>
<a href="#">Features</a>	<a href="#">Java API</a>	<a href="#">Post Mortem System</a>	<a href="#">Console Server (C)</a>	<a href="#">Quick Tutorial (UNIX/Linux)</a>	<a href="#">TINE CanOpen Manager (TICOM)</a>
<a href="#">Configuration</a>	<a href="#">Java API (ACOP)</a>	<a href="#">State Server</a>	<a href="#">Console Client (C)</a>	<a href="#">Workshop Tutorial (Buffered Server)</a>	
<a href="#">Data Types</a>	<a href="#">Buffered API</a>	<a href="#">Dialog Server</a>	<a href="#">GUI Server (VB)</a>		
<a href="#">Transfer Modes</a>	<a href="#">LabView API</a>	<a href="#">Name Server</a>	<a href="#">GUI Client (VB)</a>		
<a href="#">Access Flags</a>	<a href="#">MatLab API</a>	<a href="#">Remote Services</a>	<a href="#">Console Client (Java)</a>		
<a href="#">Array Types</a>	<a href="#">CDI Native API</a>	<a href="#">Network Globals</a>	<a href="#">GUI Client (Java)</a>		
<a href="#">Time Stamps</a>	<a href="#">Time codes</a>	<a href="#">Time Synchronization</a>	<a href="#">Console Server (Java)</a>		

TINE API: TINE API for Console Applications - Windows Internet Explorer

http://adweb.dev.de/tine/tine/cd\_bc.html

Google

TINE API: TINE API for C...

ActiveX | Visual Studio 6 + ...

Functions

```

CDI_EXPORT int cdRegisterBusCleanup (char *busName, int (*fn)(int))
registers the bus cleanup function used when CDI closes.

CDI_EXPORT int cdRegisterBusHandler (char *busName, void (*fn)(CdRequestInfoBk *))
registers the bus handler dispatch function used for bus io.

CDI_EXPORT int cdRegisterBusInitialization (char *busName, int (*fn)(int, int, char *))
registers a bus initialization function to be used by CDI when initializing the bus.

CDI_EXPORT int cdRegisterBusScanner (char *busName, int (*fn)(char *, char *, int))
registers the bus scan function used for CDI diagnostics.

CDI_EXPORT int cdRegisterCalibrationFunction (char *fnName, double (*fn)(double))
registers a calibration function for use by CDI.
    
```

Detailed Description

basic CDI library routines:

Function Documentation

```

CDI_EXPORT int cdRegisterBusCleanup ( char * busName,
int (*)(int) fn
)
    
```

registers the bus cleanup function used when CDI closes.

This routine will be called when CDI closes and allows the bus to free up resources, etc.

Parameters:

busName is the bus plug name to be used by database entries referring to this bus plug.  
fn is the address of the bus cleanup function to be called when CDI closes. The function must have the prototype int (\*fn)(int) and return 0 if the call was successful. The line parameter will contain the bus line which is being closed.

example:

```

#define BUSPLUG_NAME "SIMULATE"
#define VBUS
#define BUSPLUG_EXPORT __declspec(dllexport)
#define BUSPLUG_EXPORT
#define
BUSPLUG_EXPORT void simulationHandler(CdRequestInfoBk *pReq)
{
// handle the request ....
    
```

TINE API: TINE API for Console Applications - Windows Internet Explorer

http://adweb.dev.de/tine/tine/cd\_document.html

Google

TINE API: TINE API for C...

ActiveX | Visual Studio 6 + ...

Main Page | Features | Central Services | csv Files | Types | Transfer | Access | API-C | API-VB/ActiveX | API-Java | Examples | Downloads

## The Common Device Interface (CDI)

(Click here for documentation in German).

Introduction

The access and control of hardware devices is typically achieved via fundamental 'Get' and 'Set' operations, where a 'Get' is used to acquire data or status information from the hardware bus and a 'Set' is used to change control modes or download data to the hardware. The details behind these simple operations are in general quite varied for disparate bus types. Some bus drivers offer single-channel read and write calls, while others utilize duplex channels for read and write. Some bus drivers are single master, others are multi-master. The bus data format can also be different. For instance, RS232 deals with character string data, whereas SSI/SSD deals with short integers. The Can bus can deal with integers of various sizes. Hence the interfaces to these 'Get' and 'Set' operations are generally just as varied as the details behind them.

As all TINE developers are familiar with the TINE client API for accessing data from device servers, CDI strives to leverage this knowledge by offering the same API for accessing data from the hardware bus. In this case a device server running on a Front End Computer (FEC) is a client to its attached hardware. CDI itself offers a CDI native API which is TINE-similar. However, by and large 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.

Using the TINE client API to access the local hardware becomes a simple matter if the endpoint uses the device context "localhost", and the device server "cdi". Special parsing of the full device name also allows multiple endpoints with a single call. For instance, a call to "/localhost/cdi/device1" would access only the CDI device registered as "device1". On the other hand a call to "/localhost/cdi/device1 - device100" or "/localhost/cdi/device1,device2 - device10,device99" would identify the individual registered devices and access them as a group. The CDI property space includes the properties "RECV" for receiving (reading) data from the device, "SEND" for sending (writing) data to the device, "RECV\_RECVAUTOM" and "SEND\_RECVAUTOM" for issuing a pair-wise read-write or write-read operation which is guaranteed to be atomic, "RECV\_CLBK" and "SEND\_RECVC\_LBK" for returning data which has been calibrated according to the registered calibration rules, and such properties as "ADDRK" and "BUSNAME" which return information about the endpoint device. Other bus action properties are also available.

CDI Initialization

CDI operates on a plug-and-play basis, and 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. On windows for instance this will be cd32.dll (or cd64.dll) and on Unix systems libcd.so. Application platforms such as Java, VB, or LabView will also access this same shared library. When the library loads, it will look for a CDI bus manifest file, which is a simple comma-separated-value file and can be a simple as a single column with a list of bus plug libraries, as shown below in figure 1.

```

LIBRARY
CanEsd
cdiSsdSsb
TwinCATada
    
```

CDI will then call cdLoadLib() for each bus plug entry in the manifest list, for instance

```

cdLoadLib("cdCanEsd.dll");
    
```

on windows or

```

cdLoadLib("libcdCanEsd.so");
    
```

on Unix platforms, etc. If the library loads successfully, it will (via its prologue code) register its name and all of its bus handlers with CDI, which itself has no a priori knowledge of any hardware bus interface. After the manifest has been read, CDI will look for a CDI device database and, if found, read it and register all devices and device information contained

Local Intranet 100%