CDI Dokumentation
– Vorläufige Version für SEDPC –

DESY - MCS1

Hamburg, den 23. April 2008

Inhaltsverzeichnis

1 Prolog

Im Folgenden werden hier die beiden wichtigsten Tabellen für die Nutzung des **Common Device Interface** (CDI) erklärt und beschrieben. Dabei handelt es sich um Tabellen, die im **Comma Separated Values** (CSV) Format vorliegen müssen. Damit ist gleichzeitig auch implizit die Extension beider Dateien vorgegeben: `CSV`. Der **Ort**, an dem sich die beiden folgenden Dateien bzw. Tabellen befinden, wird in den Systemvariablen `CDI_HOME`, `FEC_HOME` oder `TINE_HOME` festgelegt. Es wird die Existenz aller drei Systemvariablen nacheinander geprüft. Sind alle drei Variablen nicht vorhanden, so müssen beide Dateien sich in dem Ordner befinden, in dem sich auch die Applikation befindet. Es wird empfohlen die Systemvariable `CDI_HOME` zu setzen.

Die betrachteten Meta-Informationen umfassen Spalten, die entweder **notwendig** oder **optional** sind. Da in der Tabelle nicht ausreichend Platz vorhanden ist, werden folgende Formulierungen vorgenommen: `MUSS` für notwendige und `KANN` für optionale Spalte.

Per Default wird immer eine **Logging-Datei** im Verzeichnis der gestarteten Applikation abgelegt. Sie lautet `cdi.log`. Es existieren momentan noch keine Optionen, die beim starten der Applikation angegeben werden können. Dieses wird sich für zukünftige Versionen von CDI ändern. Die Begriffe **Tabelle**, **Datei** und **Datenbank** werden hier als Synonyme gesehen und verwendet.

Des Weiteren werden hier die folgenden Notationen für die Metadaten definiert:

1. `<must>`: Das `must` muss angegeben werden. Z.B. für die Notation `<create>:<subadr.>` kann `1:2` oder `1:10` angegeben werden aber nicht `2` oder `2:3:4`.

2. `<must>`*`[:can]`*: Das `must` muss angegeben werden, das `can` ist optional. Wenn der optionale Anteil angegeben wird, so muss er durch einen Doppelpunkt vom `must` getrennt werden. Er ist darüberhinaus in kursiv und von eckige Klammern umgeben. Als Beispiel ist die Notation `<crate>.<basissubadr.>`*`[:<offread>[:<offwrite>]]`* für die folgenden Ausdrücke gültig: `1.2`, `1.2:10` oder `1.2:10:2`. Nicht gültig sind `1` oder `1.3:10:2-32`.

2 Applikation

2.1 Optionale Parameter

Die Applikation kann mit optionalen Parametern unter Windows XP[1] gestartet werden. Es ist dabei irrelevant, ob kein, ein oder mehrere Parameter angeben werden. Bei Mehrfachnennung eines Parameters zählt der Erste der Auftritt.[2] Wird keiner der Parameter angegeben, so werden die Default-Einstellungen verwendet. Die Eingabe von `CdiHdwSrv /help` gibt die möglichen Parameter und die Default-Einstellungen aus:

[1] Diese Parameter wurden nur unter Windows XP festgestellt. Sie sind aber sicherlich auch für weitere Betriebsysteme relevant, z.B. Linux oder Unix.

[2] FIXME: Überprüfen!

Beispiel 1

```
Usage:

        cdiHdwSrv [/n=<fecname> /c=<context> /p=<port offset> /d=<cdi home directory>]

Calling cdiHdwSrv without arguments uses the default settings.
These are: fecname = <COMPUTERNAME>.30
           context = HARDWARE
           port offset = 30
           cdi home directory = <CDI_HOME> or <FEC_HOME>
To override these use any or all of the swithes above.

For Example:

        cdiHdwSrv /n=ACCXPDOBPM /c=DORIS /p=41 /d=L:\cdi\ACCXPDOBPM
```

Damit ist es möglich mehrere CDI Server auf dem gleichen Rechnersystem zu starten.

2.2 CDI Sonderzeichen

Es werden hier des Öfteren **String-Ausdrücke** verwendet. Diese werden im Folgenden immer als Charakterausdrücke mit den folgenden Sonderzeichen verstanden. Der Stringausdruck beginnt mit einem Buchstaben und nachfolgenden Buchstaben oder Ziffern. Als einziges Sonderzeichen sind der Unterstrich ('_'), der Punkt ('.') und das Trennsymbol ('-') erlaubt. Über das Zeichen '#' wird ein **Kommentar** in der Zeile eingefügt. Dieser kann auch am Ende einer Zeile verwendet werden. Es ist darauf zu achten, dass die gesamte Zeile 256 Zeichen nicht überschreiten darf.[3]

Die folgenden Sonderzeichen werden von TINE verwendet (insbesondere bei der Softwareentwicklung unter TINE):

- ':' - CDI Tokenizer
- '/' - Hierarchischer Begrenzungszeichen
- '*<space>*' - Leerzeichen, sollte niemals in einem Device oder etwas anderem verwendet werden
- '*<space>*-*<space>*' - Bereich
- '#' - Nummer

[3] Die Länge von 256 Zeichen gilt hier nur fürs Erste. Eventuell kann eine Zeile auch länger als 256 Zeichen sein, nur wird hier vom "worst case" ausgegangen.

3 Datenbank

3.1 Die 'CDI Manifest' Datenbank

Der Name dieser Tabelle lautet `cdimf.csv`. Diese Tabelle besteht aus zwei Spalten: `LIBRARY` und `BUS_ENV`. Die erste Spalten ist notwendig, die Zweite ist optional.[4] Die Metadaten werden in der Tabelle 1 definiert.

Für die Handhabung eines Simulationsservers siehe unter Abschnitt 3.5.

Spaltenname	Verwendung	Beschreibung
`LIBRARY`	`MUSS`	Beinhaltet den Dateinamen der verwendeten Bibliothek. Es wird nur der Prefix des Dateinamens angegeben ohne Extension. Unter Windows wird hier als Extension i.d.R. ein `DLL` (Dynamic Link Library) sowie unter Linux ein `SO` (Shared Objects) angehängt. Siehe Beispiel 1.1. Daneben ermöglicht diese Spalte die Deklaration von komplizierten Berechnungsformeln, in dem in einer Bibliothek Funktionen bereit gestellt werden. Weitere Informationen gibt es hierzu im Abschnitt 3.2.3. Darüber hinaus existiert für den SEDAC Anschluß ebenfalls ein Simulationsserver mit der Bezeichnung `cdiSedPcSimul`. Siehe Beispiel 1.2.
`BUS_ENV`	`KANN`	Ist die Kennung des Bussystems oder Bus-Plugin's als String-Ausdruck.[5] Diese Bezeichnung ist abhängig von der Library.[6]

Tabelle 1: CDI Manifest Datenbank `cdimf.csv`

Beispiel 1.1: Mehrere Busse

```
1 LIBRARY,  BUS_ENV
2 cdiSedPc, SEDPC=1:378
3 cdiCanPc, CAN
```

Beispiel 1.2: Simulationsbus

```
1 LIBRARY,       BUS_ENV
2 cdiSedPcSimul, SEDPC=1:378
```

[4] Zwar ist die zweite Spalte optional, doch sollte sie immer mit angegeben werden.

[5] Diese Kennung wird für den SEDAC-Bus durch zwei weitere Zifferen charakterisiert, welche durch einen Minuszeichen getrennt sind. Zwischen der Busbezeichnung und den Ziffern wird ein Doppelpunkt angegeben. Beispiel: `SEDPC:1-378`

[6] Unter der Bibliothek `LIBRARY` werden ein bis mehrere Kennungen explizit genannt, wobei diese konkret in dieser Bibliothek als Klartext verwendet werden. Diese entsprechen den hier verwendeten Bezeichnungen des Bussystems.

3.2 Die 'CDI Address' Datenbank

Der Name dieser Tabelle lautet `cdiaddr.csv`. Diese Tabelle besteht aus maximal 16 Spalten. Davon sind die Spalten `BUS`, `LINE`, `ADDRESS` und `NAME` notwendig und `NUMBER`, `MASK`, `ACCESS`, `FORMAT`, `PATTERN`, `TOLERANCE`, `INPUT`, `LIMIT`, `RULE_RECV`, `RULE_SEND`, `DESCRIPTION` und `LONG_NAME` optional. Die Metadaten werden in der Tabelle 2 definiert.

Spaltenname	Verwendung	Beschreibung
`BUS`	`MUSS`	Diese Spalte entspricht der Spalte `BUS_ENV` der Manifest-Datenbank `cdimf.csv`. Es handelt sich um einen String-Ausdruck.[7] Die nachfolgende Definition eines Indexes muss für jeden Bus fortlaufend und eindeutig sein (siehe Beispiel 1 und 2), falls sie angegeben wird. Des Weiteren kann hier aber auch der Begriff `TEMPLATE` verwendet werden, wobei hier ein Muster beschrieben wird. Für die Defintion eines Templates sollte im Abschnitt 3.2.5 nachgelesen werden. Diese Kennung entspricht der verwendeten "Deklaration" der Bibliothek aus der Manifest-Datei und ist damit ein indirekter Hinweis auf das zu verwendende Bus-Plugin. Diese Kennung hat **keinen** Bezug zur Darstellung in der Manifest-Datei. Vielmehr verweisen beide Einträge auf die Bibliothek selbst und nicht von diesem Eintrag zur Manifest-Datei und diese wiederum auf die Bibliothek.
`LINE`	`MUSS`	Physikalische Line des jeweiligen Bussystems. Die `LINE` ist von der Hardware abhängig. Die hier angegeben Ziffer ist immer größer oder gleich der 1. Pro angegeben `BUS` können hier auch gleiche Ziffern angegeben werden. Dabei sind diese aber wiedrum unabhängig voneinander (siehe Beispiel 1).

[7] Für die Kennung des SEDAC Busses folgen nach der Bezeichnung einer Ziffernfolge (zwei Ziffern), welche über ein Gleichheitszeichen mit dem Bus verbunden wird. Beide Zahlen werden über einen Doppelpunkt getrennt.

ADDRESS	MUSS	Ist abhängig vom Bussystem. Für das Bussystem **SEDAC** wird die folgende Syntax verwendet: <crate>.<basissubaddress>[*:<offread>[:<offwrite1>[:<offwrite2>]]*]. Es wird optional zur Basis-Subadresse die Zahl offread und offwrite1/offwrite2 hinzu addiert, wobei die Erste zum lesen und die Zweite/Dritte zum schreiben verwendet wird. Wird nur ein Offset angegeben, so wird sowohl das Lesen als auch das Schreiben der gleiche Offset verwendet.
NUMBER	KANN	Ist eine eindeutige Nummer, die bei der 1 beginnend angewendet wird. Diese Nummer muss aber nicht fortlaufend sein. Die Nummer entspricht der Device-Nummer. Anschließend können über die Applikation entweder über den Namen oder die Nummer auf die Device zugegriefen werden.[8] Siehe hierzu auch im Abschnitt 3.3. Bei der Verwendung eines Templates müssen wir mit der Vergabe der Nummern aber Vorsicht walten lassen. Für weitere Informationen sollte im Abschnitt 3.2.5 nachgelesen werden.
NAME	MUSS	Ist der Name der Device, wobei diese eindeutig sein muss. Es kann entweder über den Namen oder mit der Nummer auf das Device zugegriffen werden.[9] Siehe hierzu auch im Abschnitt 3.3. Die Länge ist zurzeit auf 15 Zeichen begrenzt.[10]
MASK	KANN	Die Daten werden maskiert. Die Maske ist nur für **gelesene** Daten relevant. Geschrieben wird immer über das gesamte Format, ohne das die Maske angewendet wird. Siehe hierzu auch in der Spalte FORMAT).

[8] Zugegriffen wird auf das Device über einen Link. Dabei wird zunächst aus der Datei fecid.csv der Device-Context und der Device-Server ermittelt und anschließend die Nummer angehängt: /<device context>/<device server>/#<number>

[9] Zugegriffen wird auf das Device über einen Link. Dabei wird zunächst aus der Datei fecid.csv der Device-Context und der Device-Server ermittelt und anschließend der Name angehängt: /<device context>/<device server>/<name>.

[10] Ab der Version 4 ist die Länge auf 512 Zeichen erweitert.

FORMAT	KANN	Das Format ist Busabhängig. Für SEDAC existieren die folgenden Formate: short (16 Bit), int (32 Bit), float (32 Bit), double (64 Bit), char (8 Bit), byte (8 Bit), name32 (32 Byte). Bei den Gleitkommazahlen handelt es sich um Zahlen, die nach der IEEE 754 deklariert werden.
PATTERN	KANN	(Vorläufig) Ist abhängig vom Format. Vergleicht die gelesen Daten mit dem gespeicherten Pattern. Wenn dieser identisch oder nicht identisch mit dem gelesen Daten sind, so wird der Lesevorgang abgeschlossen, ansonsten nicht. Das Pattern ist nur für das Einlesen der Wert zuständig und ist abhängig vom Format. Siehe FORMAT für weitere Informationen.
TOLERANCE	KANN	(Vorläufig) Ist die Toleranz der Daten. Es existieren zwei mögliche Zugriffsvarianten: **absolut** oder **prozentual**. Dabei wird ein Callback-Funktion aufgerufen, falls die Grenze über der Toleranz liegt. Wird nur für den lesenden Zugriff verwendet. Die Notation für absolute Daten lautet <absolute data>, wobei gilt, dass die Daten innerhalb der absoluten Grenze liegen: $\lvert actualdata - lastdata \rvert > absolutedata$. Die Notation für prozentuale Daten lautet <procentual data>:1, wobei gilt, dass die Daten innhalb der prozentualen Grenze liegen: $\lvert \frac{actualdata - lastdata}{lastdata} \rvert > procentualdata$. Dabei sind alle Formate, bis auf die prozentualen Daten, abhängig von der Spalte FORMAT.
INPUT	KANN	Diese Spalte ist für kombinierte Schreibzugriffe gedacht: WRRD, WRWR und WRRDWR. Allen Zugriffen ist gemein, dass sie zunächst schreibend auf die Hardware zurückgreifen. Der Inalt wird hier von dieser Spalte der Hardware übergeben, bevor eine der drei anderen Operationen ausgeführt wird (in diesem Fall sind dieses RD, WR und RDWR).[11] Diese Spalte ist äußerst hilfreich für Operationen auf Sedac-Module, die intern eine erweiterten Registerauswahl besitzen, gegenüber den normalen 16 Registern.

[11] Ist sehr hilfreich u.a. für den Instant Client, der z.B. über den Zugriff WRRD zunächst den Inhalt dieser Spalte an eine bestimmte Stelle schreibt und anschließend das Ergebnis liefert. Es kann sich auch um ein internes Register handeln aus dem gelesen wird, z.B. für die Petra 3 Temperaturen.

LIMIT	KANN	Bezeichnet die Anzahl der maximal zu lesenden und zu schreibenden Daten. Die folgende Notation wird verwendet: <limit1>*[:<limit2>]* ... . Die Werte beziehen sich auf die jeweils vorhandenen Zugriffsoperationen der Spalte ACCESS.
ACCESS	KANN	Diese Spalte regelt den Zugriff auf die Devices. Es gibt mehrere Zugriffsmöglichkeiten zwischen Soft- und Hardware: RD, WR, RDWR und WRRD, WRWR und WRRDWR. Es können mehrere Zugriffsmöglichkeiten über das | verknüpft werden. Der Zugriff erfolgt **atomar**, das heißt, dass keine andere Operationen während des Zugriffs auf das Register ausgeführt werden dürfen.[12] Wird diese Zeile nicht angegeben, so ist sie für den Schreib- als auch Lesezugriff zugelassen. Es sollte darauf geachtet werden, dass wenn mehrere CDI Zeilen über TINE gemeinsam in dem Zugriffsnamen vereinbart werden (z.B. /TEST/PT100.-1.CDI/#1,#3-#7), dass alle CDI Zeilen den gleichen Zugriff benötigen um diese als Paket zu versenden. Der Zugriff aller Nummern (#1, #2, #3 und #7) ist nicht atomar. Das bedeutet, das hier z.B. die Register #1 und #2 ausgelesen, sich zwischendurch Register #3 ändert und anschließend ausgelesen wird. Die **einzelnen** Zugriffe sind aber dennoch atomar.
DESCRIPTION	KANN	Ist eine kleine Beschreibung des Devices. Diese kann bis zu 63 Zeichen lang sein und wird unter TINE als NAME64 zurückgegeben. Für weitere Informatione zu dieser Spalte siehe in dem Abschnitt 3.2.2.
LONG_NAME	KANN	Ist eine ausführlichere Beschreibung des Namens. Diese kann bis zu 63 Zeichen lang sein. **Diese Spalte sollte vom Entwickler nicht verwendet werden und ist für weitere Bus-Plugin's reserviert.**

[12] Ist das nur für den schreibenden Zugriff oder für beide relvant, also lesend und schreibend?

RULE_SEND	KANN	Diese Spalte umfaßt ebenfalls mathematische Ausdrücke, über welche Daten vom selbst geschriebenen Server an den CDI Server weitergeleitet und umgerechnet werden. Es können hier die selben mathematischen Ausdrücke wie in der Spalte RULE_RECV verwendet werden.[13]
RULE_RECV	KANN	Diese Spalte umfaßt einfache mathematische Ausdrücke, über die die Daten berechnet werden. Diese Spalte dient dem Empfangen von Daten in den selbst geschriebenen Client. Getrennt werden die einzelnen mathematischen Operationen über das Sonderzeichen ':'. Neben den einfachen mathematischen Operationen (+, -, *, /) sind auch das Potenzieren (^) und das logische Verschieben erlaubt (< und >). Darüber hinaus kann über die Operation XOR eine XOR-Operation durchgeführt werden.[14] Neben der Berechnung kann auch ein Stringausdruck für ein Bit eines Ausdrucks zurückgegeben werden: die s.g. Message. Dazu wird in dieser Spalte durch ein vorrangestelltes 'MSG' gefolgt von einen oder zwei Stringausdrücken angewendet. Damit wird also die folgende Syntax verwendet: MSG<bit value><string>[<*string*>]. Diese Spalte ist nur für den lesenden Zugriff relevant.[15] Beide Stringausdrücke werden von Guillemets[16] [17] umschlossen. Zugegriffen wird auf die Daten über die Property RECV.CLBR. In der Applikation werden die Daten über den TINE-Datentypen NAME32I zurückgegeben.[18] Daneben ermöglicht diese Spalte aber auch den Zugriff auf komplexere Funktionen, die in der Programmiersprache C umgesetzt werden. Für weitere Informationen siehe im Abschnitt 3.2.3.

Tabelle 2: CDI Adress Datenbank cdiaddr.csv

[13] Phil: können hier tatsächlich alle Ausdrücke verwendet werden, oder gibt es Einschränkungen? Zum Beispiel das verschieben der Operation?

[14] Die Bezeichnung 'XOR' kann auch auf einen Buchstaben verkürzt werden: 'X'.

[15] Die Bezeichnung 'MSG' kann auch auf einen Buchstaben verkürzt werden: 'M'.

[16] Bei einem Guillemet handelt es sich um die französischen Anführungzeichen.

[17] Kann hier etwas verwirrend wirken, den die Guillemets werden hier auch in der Definition des Ausdrucks verwendet. Das heißt aber nichts anderes, als das z.B. in der konkret angewendeten Form dem Buchstaben MSG eine Ziffer mit Stringausdrücken in Guillemets folgen, also: MSG64<RICHTIG><FALSCH>.

[18] Zurzeit ist es nicht möglich, dass ein Bereich übergeben wird.

Im Folgenden werden hier einige Beispiele und weitere Hinweise zu den in der Tabelle 2 ausgeführten Spalten genannt:

`MASK`: Z.B. wird über die Maske `0x3fff` die ersten zehn Bits ausgelesen, aber die Bits von 11 bis 15 ignoriert, falls es sich um eine `short`-Format handelt. Die Spalte `MASK` wird **vor** der Spalte `RULE_RECV` abgearbeitet.

`LONG_NAME` Wird für das `TWINCAT` Bussystem oder Bus-Plugin verwendet.

`DESCRIPTION` Frage: *ist diese Spalte "case sensitive"? Wobei sich hier die Frage stellt, ob diese Spalte von einer Maschine automatisch abgearbeitet werden muss, oder ob sie nur für den Menschen lesbar ist. Wenn letzteres der Fall ist, ist es auch egal ob sie case sensitive ist oder nicht.*

`RULE_RECV` Für die Verwendung einer Entscheidung über den `MSG`-Ausdruck. Es gibt drei mögliche Fälle:

1. Es ist kein Ausdruck angegeben. Somit wird einfach der zurückgegeben Wert von der Hardware weitergereicht (evtl. unter Anwendung der Maske `MASK` etc.).
2. Es ist ein Wert mit einem Stringausdruck in Guillemets angegeben. In diesem Fall wird überprüft, ob der von der Hardware gelieferte Wert mit dem in der Spalte übereinstimmt. Ist dieses der Fall wird der Stringausdruck weiter gereich ansonsten wird ein Leerstring zurückgegeben.
3. Es ist der Wert mit zwei Stringausdrücken vereinbart, wobei beide Stringausdrücke in Guillemets angegeben werden. In diesem Fall wird überprüft, ob der von der Hardware gelieferte Wert mit dem in der Spalte übereinstimmt. Ist dieses der Fall wird der erste Stringausdruck weitergereicht, ansonsten der Zweite. Siehe Beispiel 2.

Diese Spalte wird in der **Hochfrequenze** für Fehler- und Statusmeldungen verwendet. Umgesetzt wird hier ein Ausdruck, der entweder wahr oder falsch ist. Als Beispiele sei hier z.B. die folgenden Zeilen genannt: `MSG4<EIN>`, `MSG8<><EIN>` und `MSG8<EIN><AUS>`. Die erste Zeile gibt, wenn der Ausdruck einen `4` liefert, den String-Ausdruck `EIN` zurück, ansonsten ein Leerstring. Das gleiche gilt in der darauffolgende Zeile, nur wird hier für den **nicht gesetzten** Wert von `8` der Ausdruck `EIN` zurückgegeben. Die letzte Zeile gibt wieder für den Ausdruck `8` den String-Ausdruck `EIN`, ansonsten `AUS` zurück. Als TINE-Datentyp wird hier `NAME32I` verwendet, wobei der Stringausdruck die Nachricht enthält und einen Integerwert, der evtl. den mitgesendeten Fehlercode enthält (z.B. SEDAC-Fehler). Ist dieser ungleich der `0` so handelt es sich um einen Fehler, ansonsten ist der Wert O.K. Vergleiche hierzu mit dem Abschnitt 3.2.4.

`ACCESS` Es können auch Bitwerte manipuliert werden. Dieses geschieht über einen `RDWR` und `WRRDWR` Zugriff.[19] Gesendet wird ein Bytewert gefolgt von einem Bitwert. Beide werden durch ein **Leerzeichen** getrennt: `<bytevalue>␣<bit>`. Der Ausdruck

[19] Oder unter dem Instant Client über einen `SEND.RECV.ATOM` bzw. `SEND.RECV.SEND.ATOM` Zugriff

<bytevalue> richtet sich nach dem verwendeten FORMAT. Das bit wird entweder gelöscht (0) oder gesetzt (1).[20]

Als Beispiel wird hier der zu ändernde 16-Bit Registerinhalt 1001 1000 1100 1011 verwendet. Wird hierauf die Änderung 6␣1 angegeben, ergibt sich 1001 1000 1100 1111. Wird anders herum die Änderung 6␣0 angegeben, ergibt sich 1001 1000 1100 1001.

Für das Versenden als Pakete müssen alle CDI Zeilen in einem TINE Aufruf den gleichen Zugriff aufweisen. Unterscheiden sich einer oder mehrere von ihnen, so wird dieser in mehrere Pakete unterteilt und gesendet.

3.2.1 Weitere Beispiel für die CDI-Tabelle

Beispiel 1

```
1 BUS        , LINE, ADDRESS , NUMBER, NAME          , ACCESS, FORMAT, LIMIT
2 SEDPC=1:378, 1   , 1.16:1:0, 10    , HETemGrp1-8   , RD    , short , 08:01
3 SEDPC      , 1   , 1.32:1:0, 11    , HETemGrp9-16  , RD    , short , 08:01
4 SEDPC      , 1   , 1.64:1:0, 12    , HETemGrp17-24 , RD    , short , 08:01
5 CAN        , 1   , 1.32    , 20    , Resistor1     , RD    , short ,
6 CAN        , 1   , 1.32    , 21    , Resistor2     , RD    , short ,
7 ...
```

Beispiel 2: Die Spalte RULE_RECV mit Message

```
1 BUS        , ... , MASK  , ACCESS, FORMAT, RULE_RECV
2 SEDPC=1:378, ... , 0x0080, RD    , short , MSG4<EIN><AUS>
3 SEDPC      , ... , 0xffff, RD    , short ,
4 SEDPC      , ... , 0x0004, RD    , short , MSG8<Wasserruecklauf defekt>
5 ...
```

Das folgende Beispiel zeigt die Operation des XOR der Spalte RULE_RECV. Wir gehen hier davon aus, dass ein Hardwareentwickler sich für das 6 Bit der Adresse 1.16 den Bitwert 1 der Parameter gesetzt ist (EIN) und für den Bitwert 0 dieser nicht gesetzt ist (AUS):

Beispiel 3a: Die Spalte RULE_RECV ohne XOR

```
1 BUS  , ... , ADDRESS , MASK  , ACCESS, FORMAT, RULE_RECV
2 SEDPC, ... , 1.16    , 0x0020, RD    , short , MSG32<EIN><AUS>
3 ...
```

Nach längerer Zeit stellt er seinen Irrtum fest und möchte das genaue Gegenteilige realisieren. Wir können nun entweder die entsprechende Zeile die Ausgabe der Nachricht ändern (also von MSG32<AUS><EIN>), was unter Umständen sehr aufwendig sein kann, oder wir führen die XOR Operation in die CDI-Tabelle ein. Sie dreht das gewünschte Ergebnis einfach um.

Beispiel 3b: Die Spalte RULE_RECV mit XOR

```
1 BUS  , ... , ADDRESS , MASK  , ACCESS, FORMAT, RULE_RECV
2 SEDPC, ... , 1.16    , 0x0020, RD    , short , XOR 32:MSG32<EIN><AUS>
3 ...
```

[20] HIER FEHLEN BEISPIELE! WIE SIEHT DIESE SPALTE Z.B. AUS, WENN ICH LEDIGLICH DAS 6. BIT AUF 1 SETZEN WILL: WR 128 1.?

3.2.2 Merkmale der Spalte DESCRIPTION

Die Spalte DESCRIPTION kann neben einer kurzen Beschreibung der Zeile auch besondere Merkmale einer Device aufweisen.[21] Momentan existieren drei Merkmale, die in eckigen Klammern zusätzlich angegeben werden können. Dabei werden die einzelnen Merkmale über ein Leerzeichen voneinander getrennt. Alle diese Merkmale haben **keine** Auswirkung auf die über CDI bzw. TINE zurück gelieferten Daten, sondern es sind lediglich Informationen über das Device selbst.

Zu den drei Merkmalen gehören die Zahlengenauigkeit (`precision`), die Einheit (`units`) und eine Wertebereich (`range`). Alle diese Merkmale werden über den CDI-Server unter TINE über die entsprechenden Properties `PRECISION`, `UNITS` und `RANGE` zur Verfügung gestellt und können ausgelesen werden.[22] Die Darstellung der Merkmale in der CDI-Tabelle kann auf einen einzigen Buchstaben abgekürzt werden. Also wird aus dem Begriff `precision` ein einfaches `p`. Die Darstellungsinformation wird mit dem Wert über ein Gleichheitszeichen verbunden.

- **Zahlengenauigkeit** Als Formatierung der Genaugigkeit wird der Vor- und Nachkommabereich, beide getrennt durch einen Punkt, angegeben: `precision=width.decimal`. Z.B. bezeichnet die Darstellung `precision=6.4` einen freizuhaltenden Vorkommabereich von 6 Ziffern und einen Nachkommabereich von 4 Ziffern. Zurückgegeben wird dieses Merkmal über die TINE-Property `PRECISION` mit den TINE-Datentypen `NAME16`.

- **Einheit** Die Einheit wird als freier String-Ausdruck, ohne Leerzeichen definiert: `units=units`. Z.B. definiert die Einheit `A` der Ausdruck `units=A`. Zurückgegeben wird das Merkmal über die TINE-Property `UNITS` mit den TINE-Datentypen `NAME16`.

- **Wertebereich** Als Wertebereich wird die untere und obere Grenze zurückgegeben, beide über einen Doppelpunkt getrennt. Als Synonym wird hin und wieder auch der Begriff der minimalen und maximalen Grenze verwendet. Z.B. gibt der Wertebereich `range=0.0:321.291` als untere Grenze die Zahl 0.0 und als obere Grenze die Zahl 321.291 zurück. Beide Werte werden als Merkmal über die TINE-Property `RANGE` mit den TINE-Datentypen `FLOAT` zurückgegeben.

Als Beispiel sei hier der folgende Tabelle angegeben:

Ein Beispiel für die DECLARATION

```
1 BUS         , NAME       , ..., DECLARATION
2 SEDPC=1:378, Temp1-8    , ..., [precision=4.2 units=Grad range=0:100] Temperature
3 SEDPC       , TempMin1-8, ..., [precision=4.2 units=Grad range=0:100] Temp. min.
4 SEDPC       , TempMax1-8, ..., [precision=4.2 units=Grad range=0:100] Temp. min.
```

[21] Zurzeit ist es leider nicht möglich, dass der Bereich eines Merkmals über das Verbindungszeichen '-' als Bereich ordentlich übergeben wird.

[22] Leider ist es zurzeit nicht möglich (z.B.) über das `UNITS` Merkmal mehrerer Devices abzurufen. Dieses führt unter TINE zu einer fehlerhaften Rückgabe. Wird nur ein einzelnes Device abgefragt, so wird das Property richtig übergeben.

```
SEDPC      , Voltage   , ..., [precision=3.3 units=V] Voltage
...
```

Der Ausdruck innerhalb der eckigen Klammern ist unsichtbar für das Device und wird damit nicht berücksichtigt.

3.2.3 Funktionen unter der Spalte `RULE_RECV` und `RULE_SEND`

In der CDI Manifest-Datei kann, genauso wie ein Bussystem oder Bus-Plugin, in der Spalte `LIBRARY` eine Kalibrirungsbibliothek registriert werden.[23] In dieser Bibliothek werden eine oder mehrere Kalibrirungsfunktionen definiert, die den folgenden Prototypen in der Programmiersprache C definieren:

```
double myFunction(double value_)
```

Der Funktionsname `myFunction` kann frei gewählt werden, sollte jedoch nicht mit anderen Funktion in Konflikt treten. Diese Funktion kann nun in der Spalte `RULE_RECV` und `RULE_SEND` nach der Registrierung verwendet werden, in dem dort einfach der Funktionsname angegeben wird.

Im folgenden Beispiel wird die Bibliothek `cdiClbrFcn.dll` verwendet, welche vier Funktionen (`bit12sgn`, `nib2exp`), sowie `bit12Recv` und `bit12Send` aufweisen. Die Defintion in der Datei `cdimf.csv` lautet nun wie folgt:

Defintion der Bibliothek in cdimf.csv

```
LIBRARY   , BUS_ENV
cdiSedPc  , SEDPC=1:378
cdiClbrFcn,
```

Die Anwendung der entsprechenden Funktion `bit12Recv` aus der Bibliothek `cdiClbrFcn` in der CDI Tabelle `cdiaddr.csv` der PIA HF sieht nun wie folgt aus:

Verwendung der Funktion in cdiaddr.csv

```
BUS   , ... , MASK  , ACCESS, FORMAT, RULE_RECV            , RULE_SEND
SEDPC , ... , 0x0fff, RD|WR , SHORT , |<bit12Recv>:*0.0977 , *10.235415:|<bit12Send>
...
```

Die momentan verwirklichten Funktionen `bit12sgn`, `nib2exp`, `bit12Recv` und `bit12Send` sind die einzigen umgesetzten. Die erste Funktion ermöglicht die Rückgabe des Wertes für die erste 12 Bits mit dem Vorzeichen an der Bit-Position 12[24] (`0` = positiv sowie `1` = negativ). Die zweite Funktion ermöglicht die Rückgabe eines Wertes, der mit den Wert der e Funktion potenziert und multipliziert wird. Der Wert wird in die unteren 12 und oberen 4 Bits unterteil. Potenziert wird mit den oberen 4 Bits um das Ergebnis mit den unteren 12 Bit zu multiplizieren. Die dritte und vierte Funktion entspricht weitesgehend der ersten Funktion bis auf die Tatsache, das bei `bit12Recv` die Zahl 2047 subtrahiert und bei `bit12Send` die Zahl 2047 addiert wird.

[23] Siehe hierzu im Abschnitt 3.1 nach.

[24] Es wird hier von einer Durchzählung der Bits ab Null ausgegangen.

Das verwendete Pipe-Symbol | (es kann auch das F Symbol verwendete werden) wird für die registrierte Kalibrirungsfunktion verwendet. Ist die Funktion nicht in der Bibliothek als Prototyp definiert, so wird der entsprechende Aufruf der Spalte RULE_RECV bzw. RULE_SEND ignoriert.

Hier ist als Beispiel der Funktioneninhalt der beiden Funktionen bit12Recv und bit12Send angegeben:

Funktionen bit12Recv und bit12Send

```
...
CDICLBRFCN_EXPORT double bit12Recv(double value)
{
   short current = (short) value;
   current &= 0x07ff; // only get 11 bit
   current -= 2047;
   return (double) current;
}

CDICLBRFCN_EXPORT double bit12Send(double value)
{
   short current = (short) value;
   current += 2047;
   return (double) current;
}
...
```

Zu finden ist der Quelltext unter dem folgenden Pfad: Z:/projects/Service/vc++/-CommonDeviceInterface/busPlugs/Platform/Win32/cdiClbrFcn.

3.2.4 Übergebene Datentypen unter TINE, Fehlercode

Bei den übergebenen Datentypen unter dem TINE Interface kann es sich immer um einen zwei oder mehrfachen Datentypen handeln. Der letzte ist immer der Datentyp INTEGER, welcher für eine Fehlerbehandlung verwendet wird. Ist dieser ungleich der Null, so liegt ein Fehler vor. Die Fehlernummer wird über die Klassenmethode de.desy.tine-.defintions.TErrorList.getErrorString(int code) als String Ausdruck zurückgegeben. In der Tabelle 3 werden alle möglichen TINE Datentypen angegeben die einen Integer Fehlercode beinhalten.

Datentyp	Beschreibung
NAME16I	Ist ein 16 Charakter umfassender Stringausdruck mit Integer-Fehlercode.
NAME32I	Ist ein 32 Charakter umfassender Stringausdruck mit Integer-Fehlercode.
NAME48I	Ist ein 48 Charakter umfassender Stringausdruck mit Integer-Fehlercode.
NAME64I	Ist ein 64 Charakter umfassender Stringausdruck mit Integer-Fehlercode.

NAME16FI	Ist ein 16 Charakter umfassender Stringausdruck, Float-Wert mit Integer-Fehlercode.
NAME16II	Ist ein 16 Charakter umfassender Stringausdruck, Integer-Wert mit Integer-Fehlercode.
NAME32I	Ist ein 32 Charakter umfassender Stringausdruck mit Integer-Fehlercode.
NAME64I	Ist ein 64 Charakter umfassender Stringausdruck mit Integer-Fehlercode.
FLTINT	Ist ein Float-Wert mit Integer-Fehlercode.
INTFLTINT	Ist ein Integer-Float-Wert mit Integer-Fehlercode.
INTINT	Ist ein Integer-Wert mit Integer-Fehlercode.
INTINTINT	Ist ein Integer-Integer-Wert mit Integer-Fehlercode.
FFI	Ist ein Float-Float-Wert mit Integer-Fehlercode.
FII	Ist ein Float-Integer-Wert mit Integer-Fehlercode.
FIFI	Ist ein Float-Integer-Float-Wert mit Integer-Fehlercode.

Tabelle 3: TINE Datentypen mit Fehlercode

Es kann aber auch auf die Identifizierung des Fehlers verzichtet werden. In diesem Fall wird der Fehlercode als Datentyp nicht angegeben.

3.2.5 Definition eines Musters über TEMPLATE

Es gibt auch die Möglichkeit das ein Muster für ein abstraktes Gerät definiert wird um im Anschluß die konkreten Devices zu deklarieren. Diese Deklaration bezieht sich auf das vorher definierte Muster. Im konkreten Anwendungsfall wird in der Spalte BUS das Schlüsselwort TEMPLATE angegeben.

Die Spalte NAME setzt sich immer aus einen eindeutig definierten Gerät oder Device gefolgt von einer eindeutig und individuell zu wählenden Bezeichnung zusammen; beide werden durch einen Dopplepunkt getrennt:

<pattern>:<description>

Die <description> sollte keinen Sonderzeichen enthalten.[25] Als Beispiel seien hier die abstrakten Properties TEMP:TMi oder TEMP:Tmp des abstakten Gerätes TEMP genannt.

Die LINE ist immer 0. Die Spalte ADDRESS und INPUT müssen angegeben werden[26], wobei die Spalte INPUT immer den entsprechend übergebenen Wert enthält; hierzu später mehr. Die Adresse hat ebenfalls eine gesonderte Stellung in der Template Definition. Für SEDAC wird für das Crate und die Subadresse immer der Parameter 0.0 angegeben. Dieses ist ein Platzhalter für die später folgende Beschreibung der Devices. Anschließend

[25] Auch hier gilt, dass diese Bezeichnung durchaus Sonderzeichen beinhalten darf, nur leider nicht alle. Z.B. sollte der Space-Charakter vermieden werden aber der Trennstrich ('-') ist durchaus erlaubt.

[26] Wir benötigen die Spalten ADDRESS und INPUT für die nachherige konkrete Deklaration der Devices.

werden die Offsets für den oder die Zugriffe (befinden sich in der Spalte `ACCESS`) angegeben. In unserem Fall besitzen wir einen Schreib-Lese-Zugriff. Der Schreibzugriff wird im Folgenden immer auf das 0. Register durchgeführt, der Lesezugriff auf das 1. Register (in unserem Fall lautet es also `WRRD`). Identisch ist dieser Zugriff im Übrigen auch für `WRWR`, wobei das letzte Register nicht gelesen sonder geschrieben wird.[27]

Beispiel: Deklaration eines Templates

```
BUS     , ADDRESS,NAME     ,ACCESS   ,INPUT ,FORMAT,LIMIT,RULE_RECV
TEMPLATE, 0.0:1:0,TEMP:Tmp ,WRRD     ,0x8000,SHORT ,08:01,
TEMPLATE, 0.0:1:0,TEMP:TMi ,WRRD     ,0x8008,SHORT ,08:01,
TEMPLATE, 0.0:1:0,TEMP:TMa ,WRRD     ,0x8010,SHORT ,08:01,
TEMPLATE, 0.0:1:0,TEMP:5V  ,WRRD     ,0x001d,SHORT ,01:01,*0.0025:+4.25
TEMPLATE, 0.0:1:0,TEMP:IV  ,WRRD     ,0x8020,SHORT ,08:01,*0.005
TEMPLATE, 0.0:1:0,TEMP:MV  ,WRRD     ,0x8028,SHORT ,08:01,*0.005
TEMPLATE, 0.0:1:0,TEMP:Ctrl,WRRD|WRWR,0x0058,SHORT ,01:01,
...
```

Anmerkung: Petra 3, Temperaturen
Die Spalte `INPUT` beinhaltet in diesem Fall für die Temperaturen von Petra 3 teilweise einen besonderen Wert. Ist hier das oberste Bit gesetzt, so wird von der Hardware für das mehrfache Auslesen das interne Register automatisch inkrementiert. Das Bedeutet, dass z.B. für einen Input-Wert von `$8000` und einem 8-fachem Auslesen[28] das Register 0 bis 7 ausgelesen wird.

Die eigentliche Deklaration eines vorher definierten Musters geschieht über die Adressspalte. Der Adresse wird der Template- oder Musternamen durch einen Doppelpunkt angehängt und die konkrete Crate und Subadresse genannt. Anschließend wird der Templatename durch einen Doppelpunkt angehängt. Das folgende Beispiel zeigt die Deklaration des oben definierten Musters `TEMP` in einzelnen Instancen.

Beispiel: Definition eines Templates

```
BUS         , LINE, ADDRESS     , NUMBER, NAME, ...
SEDPC:1-378,    1, 1.16:<TEMP> ,      1,   H1, ...
SEDPC      ,    1, 1.32:<TEMP> ,      2,   H2, ...
SEDPC      ,    1, 1.64:<TEMP> ,      3,   H3, ...
SEDPC      ,    1, 31.32:<TEMP>,      4,   T1, ...
SEDPC      ,    1, 31.64:<TEMP>,      5,   T2, ...
```

Wir müssen jedoch bei der Defintion eines Templates mit normalen Devices Vorsicht walten lassen. Bei einer solchen Definition müssen wir auf die zu vergebenden Werte der Spalte `NUMBER` aufpassen. Es kann zu einem Konflikt bzw. zu einer Kollision zwischen der automatisch und dynamisch vergebenen Devicenummern eines Templates mit der manuellen eines normalen Devices kommen. Als Beispiel sei hier die Definition eines Templates mit einer normalen Device genannt. Bei der automatischen und dynamischen

[27] Das 0. Register entnimmt CDI der Tabelle aus der Spalte `INPUT` solange kein anderer Wert angegeben ist.
[28] Siehe hierzu in der Spalte `LIMIT`.

Nummernvergabe des Templates gehen wir von einem Algorithmus aus, der die Nummern fortlaufend ab der 10 für Devices vergibt die im Template deklariert wurden.[29]

Beispiel: Definition eines Templates

```
BUS         , LINE, ADDRESS     , NUMBER, NAME     , ACCESS,  INPUT, ...
TEMPLATE    ,    0, 0.0:1:0     ,      0, TEMP:Tmp ,   WRRD, 0x8000, ...
TEMPLATE    ,    0, 0.0:1:0     ,      0, TEMP:TMi ,   WRRD, 0x8008, ...
TEMPLATE    ,    0, 0.0:1:0     ,      0, TEMP:TMa ,   WRRD, 0x8010, ...
SEDPC:1-378,     1, 1.16:<TEMP> ,      1, H1       , ...
SEDPC       ,    1, 1.32:<TEMP> ,      2, H2       , ...
SEDPC       ,    1, 1.64:<TEMP> ,      3, H3       , ...
SEDPC       ,    1, 2.16:<TEMP> ,      4, T1       , ...
SEDPC       ,    1, 2.32:<TEMP> ,      5, T2       , ...
SEDPC       ,    1, 2.48:<TEMP> ,      6, T3       , ...
SEDPC       ,    1, 2.64:<TEMP> ,      7, T4       , ...
SEDPC       ,    1, 2.80:<TEMP> ,      8, T5       , ...
SEDPC       ,    1, 2.96:<TEMP> ,      9, T6       , ...
SEDPC       ,    1, 10.16:1:0   ,     10, VOLTAGE  , ...
```

Nach Auflösung des Templates wird die folgende Tabelle angelegt:

Beispiel: Definition eines Templates

```
BUS         , LINE, ADDRESS     , NUMBER, NAME     , ACCESS, INPUT , ...
SEDPC:1-378,     1, 1.16:1:0    ,      1, H1.Tmp   , WRRD  , 0x8000, ...
SEDPC       ,    1, 1.16:1:0    ,     10, H1.TMi   , WRRD  , 0x8008, ...
SEDPC       ,    1, 1.16:1:0    ,     11, H1.TMa   , WRRD  , 0x8010, ...
SEDPC       ,    1, 1.32:1:0    ,      2, H2.Tmp   , WRRD  , 0x8000, ...
SEDPC       ,    1, 1.32:1:0    ,     12, H2.TMa   , WRRD  , 0x8008, ...
...
SEDPC       ,    1, 10.16:1:0   ,     10, Voltage  , WRRD  , 0x0100, ...
```

In dem obigen Beispiel kommt es zu einem Konflikt mit der umgewandelten Template-Device `H1.TMi` und der Device `Voltage`. Beide würden auf die Devicenummer 10 verweisen, was nicht sein darf. Wir sollten aus diesem Grund ein möglichst hohe Nummer für die Vergabe der normalen Devices wählen. Es wird Empfohlen sich die Datei `cdi.log` anzusehen um mögliche Konflikte zu identifizieren.

3.2.6 Definition eines Bitfeldes über `BITFIELD`

Diese Umsetzung ist erst ab der Version 4 relevant, wobei hierzu sichlich noch eine kleine E-Mail von Phil folgen wird.

3.3 Namen und Nummern

Über den Namen (Spalte `NAME`) bzw. die Nummer (Spalte `NUMBER`) wird eine Device definiert. Im Programmtext gibt es nun mehrere Möglichkeiten über TINE auf einzelne

[29] Tatsächlich sieht der Algorithmus folgendermaßen aus: Zunächst wird bei der Verwendung eines Templates die Devicenummer dynamisch vergeben. Das Template erhält als ersten die angegebene Devicenummer. Ist diese bereits vergeben, so wird die Devicenummer erhöht um die Grösse der Datenbasis + 100 verwendet usw. Wenn nun "wilde" Devicenummern ausgewählt werden, kann es natürlich zu einem Konflikt kommen.

Devices zugreifen zu können. Es ist bei der Verwendung der Nummer darauf zu achten, dass diese über ein vorangestelltes `#` in einem TINE-Aufruf[30] gekennzeichnet wird. Als Device Context und Server wird im Folgenden die Bezeichnung `/TEST/PT100.1.CDI/` gewählt.

- Entweder wir bezeichnen eine einzelne Device über ihren Namen oder über ihre Nummer.

Beispiel 3: Name oder Nummer

```
1 BUS        , LINE, ADDRESS , NUMBER, NAME         , ACCESS, FORMAT, LIMIT
2 SEDPC:1-378, 1   , 1.16:1:0, 1     , HETemGrp1-8  , RD    , short , 08:01
3 ...
```

  Über das oben angeführte Beispiel kann entweder über den Context `/TEST/PT100.1.CDI/HETemGrp1-8` oder `/TEST/PT100.1.CDI/#1` die Daten ausgelesen werden.

- Über die Nummern `1` und `4` können wir auch mehrere Devices im Context über das Symbol ',' vereinen.

Beispiel 4: Mehrere Nummern

```
1 BUS        , LINE, ADDRESS , NUMBER, NAME         , ACCESS, FORMAT, LIMIT
2 SEDPC:1-378, 1   , 1.16:1:0, 1     , HETemGrp1-8  , RD    , short , 08:01
3 SEDPC      , 1   , 1.32:1:0, 2     , HETemGrp9-16 , RD    , short , 08:01
4 SEDPC      , 1   , 1.64:1:0, 3     , HETemGrp17-24, RD    , short , 08:01
5 SEDPC      , 1   , 1.16:1:0, 4     , Calib1-8     , RD    , short , 08:01
6 CAN        , 1   , 1.32    , 10    , Resistor1    , RD    , short , ...
```

  Über das oben angeführte Beispiel können wir über den Kontext `/TEST/PT100.1.CDI/#1,#3` 16 Temperaturen auslesen.

- Über die Nummern `1` bis `3` können wir auch eine Gruppe von Devices im Kontext über das Symbol '-' auslesen.

Beispiel 5: Gruppe von Nummern

```
1 BUS        , LINE, ADDRESS , NUMBER, NAME         , ACCESS, FORMAT, LIMIT
2 SEDPC:1-378, 1   , 1.16:1:0, 1     , HETemGrp1-8  , RD    , short , 08:01
3 SEDPC      , 1   , 1.32:1:0, 2     , HETemGrp9-16 , RD    , short , 08:01
4 SEDPC      , 1   , 1.64:1:0, 3     , HETemGrp17-24, RD    , short , 08:01
5 SEDPC      , 1   , 1.16:1:0, 4     , Calib1-8     , RD    , short , 08:01
6 CAN        , 1   , 1.32    , 10    , Resistor1    , RD    , short , ...
```

  Über das oben angeführte Beispiel können wir über den Kontext `/TEST/PT100.1.CDI/#1-#3` alle 24 Temperaturen auslesen. Analog erhalte ich die selben Daten über den Kontext `/TEST/PT100.1.CDI/HETemGrp1-8␣-␣HETemGrp17-24` der beiden Namen.

- Es können aber auch Kombinationen von Nummern vereint werden.

Beispiel 6: Kombinationen

```
1 BUS        , LINE, ADDRESS  , NUMBER, NAME         , ACCESS, FORMAT, LIMIT
2 SEDPC:1-378, 1   , 1.16:1:0 , 1     , HETemGrp1-8  , RD    , short , 08:01
```

[30] Siehe hierzu in der TINE API Dokumentation unter der Klasse `de.desy.tine.client.TLink`.

```
3 SEDPC      , 1   , 1.32:1:0 , 2     , HETemGrp9-16 , RD    , short , 08:01
4 SEDPC      , 1   , 1.64:1:0 , 3     , HETemGrp17-24, RD    , short , 08:01
5 SEDPC      , 1   , 1.16:1:0 , 4     , Calib1-8     , RD    , short , 08:01
6 SEDPC      , 1   , 1.128:1:0, 5     , HETemGrp25-32, RD    , short , 08:01
7 CAN        , 1   , 1.32     , 6     , Resistor1    , RD    , short , ...
```

So kann z.B. für das oben angeführte Beispiel über den Context `/TEST/PT100.1-.CDI/#1-#3,#5` alle 32 Temperaturen ausgelesen werden.

- Der Name kann **nur** für eine Gruppe von Devices über das Trennsymbol '-' definiert werden, aber nicht durch eine Anreihung von Devices (getrennt über das Komma). Es ist unbedingt darauf zu achten, dass vor und nach dem Trennsymbol ein **Leerzeichen** folgt!

Beispiel 6: Kombinationen

```
1 BUS        , LINE, ADDRESS  , NUMBER, NAME         , ACCESS, FORMAT, LIMIT
2 SEDPC:1-378, 1   , 1.16:1:0 , 1     , HETemGrp1-8  , RD    , short , 08:01
3 SEDPC      , 1   , 1.32:1:0 , 2     , HETemGrp9-16 , RD    , short , 08:01
4 SEDPC      , 1   , 1.64:1:0 , 3     , HETemGrp17-24, RD    , short , 08:01
5 ...
```

So kann z.B. für das oben angeführte Beispiel über den Sequence `/TEST/PT100.1-.CDI/HETemGrp1-8␣-␣HETemGrp17-24` alle 24 Temperaturen ausgelesen werden.

Im Programmtext kann auf eine Reihe von Devices zugegriffen werden, in dem hierzu eine Nummer verwendet wird. So kann über das Sonderzeichen `,` eine Auswahl an einzelnen Devices aneinander gereiht werden.[31]

3.4 Bezeichnung des Devicenamens

Aufgrund der ungünstigen Konstruktion von TINE und CDI, kann nur der **Devicename** von dem Entwickler frei gewählt werden um eine Device eindeutig zu bezeichnen. Dieser Name kommt direkt aus der `cdiaddr.csv` Datenbank der Spalte `NAME`.[32] Der Device Context und Device Server werden über die Tabelle `fecid.csv` festgelegt. Darüber hinaus braucht TINE den Propertynamen (z.B. für die Festlegung des Zugriffs auf die Devices. Hier befinden sich u.a. die Befehle für das Schreibende (`SEND`) oder Lesende (`RECV`) der Devices.)

3.5 Simulationsserver

Im Folgenden wird ein Simulationsserver besprochen und welche Möglichkeiten dieser einem Entwickler bietet.

Der Sinn und Zweck eines Simulationsservers besteht in der Rückgabe von vordefinierten Werten für alle Devicenamen eines bestimmten Kontexts und Servers. Nach der

[31] Dieses sollte momentan aber nur für **Nummern** verwendet werden und nicht für **Namen**, da diese meist über den 16 Zeichen Stringausdruck hinausgehen. In späteren Versionen, in denen hoffentlich nicht mehr diese Restriktion vorliegt, wird dieses aber auch für den Namen funktionieren.

[32] Neben den Namen kann hier auch die Nummer verwendet werden. Siehe hierzu im Abschnitt 3.3.

Initializierung werden für die Devicenamen ja nach Typ immer der Defaultwert zurückgegeben. Zum Beispiel wird für den TINE Datentypen `INTINT` ein `0 0` geliefert. Auch für den Datentypen `NAME16II` wird ein `0 0` geliefert, wobei ein Leerstring vorangestellt wird.

Möchten wir andere Werte zurückerhalten, so müssen wir den entsprechenden Wert vorher schreiben, um sie nachher lesen zu können. Hierzu muss aber u.U. die CDI Tabelle unter `cdiaddr.csv` unter der Spalte `ACCESS` verändert werden. Das ist meist durch die Änderung von nur lesend (`RW`) auf schreibend (`RD|WR`) machbar, hat aber den entscheidenden Nachteil, dass diese Veränderungen bei Beendigung der Simulation wieder rückgängig gemacht werden müssen. Des Weiteren muss für einen spezieller Simulationsserver ein Programm geschreiben werden, welches die ensprechenden Werte in den Devicenamen hinterlegt.

Als Variante wird hier der folgende Vorschlag für die Umsetzung der CDI Tabellen gemacht:

- Es werden zwei Dateien als CDI Tabellen unter dem aktuellen CDI Pfad geschaffen, eine mit der Endung `_normal`, die Andere mit der Endung `_simul`.
- In der normalen CDI Tabelle werden die Zugriffsrechte unter der Spalte `ACCESS` entsprechend der jeweiligen Devicename für den realen Einsatz umgesetzt. Unter der simulations CDI Tabelle werden die Zugriffsrechte alle als Lese-/Schreib-Operation `RW|WR` behandelt. U.U. ist eine entsprechende Änderung in der Spalte `RULE_SEND` umzusetzen.
- Je nachdem welchen Server ich derzeit aktiviert habe, wird die entsprechende Datei unter den Namen `cdiaddr.csv` kopiert und der CDI Server neu gestartet.

Das hat natürlich den Vorteil, dass sich Änderungen an der CDI Tabelle auf die jeweiligen Fall konzentriert und wir immer davon ausgehen können, dass die reale CDI Tabelle mit der Endung `normal` immer funktionstüchtig ist. Sie hat aber auch den Nachteil, dass damit zwei statt einer CDI Tabelle verwaltet werden müssen.

3.6 Besonderheiten: Lesende Operationen

Im Folgenden werden hier die Vor- und Endstufe (analoge Werte) der HF 10,4 des Vorbeschleunigers PIA besprochen. Leider funktioniert das Auslesen der analogen Werte nicht über den folgenden Algorithmus:

1. Schreibe einen Wert an eine bestimmte Adresse.
2. Lese einen oder mehrere Werte aus den folgenden Adressen. Die Adressen werden dabei automatisch inkrementiert.

Vielmehr muss für jeden zurückgelieferten Wert jedesmal in die Hardware ein konstanter Wert geschrieben werden, um anschließend die Daten zu erhalten:

1. Schreiben einen konstanten Wert in eine bestimmte Adresse.

2. Lesen den Wert ab einer bestimmten Adresse; CDI wird eine Offset mitgeteilt.

3. Wiederhole diesen Vorgang im ersten Punkt falls noch weitere Werte vorhanden sind.

Dieser Zugriff ermöglicht keinen Zugriff auf die Hardware "in einem Rutsch", wie im ersten Algorithmus angegeben.
In unserem Fall werden die folgenden Operationen ausgeführt:

- `..., 2, 16.112:8:0 ,ESUAnode ,WRRD,SHORT,,-22016` ⇒ schreibe in Register `16.112`, lese von Register `16.120`
- `..., 3, 16.112:9:0 ,ESIAnode ,WRRD,SHORT,,-22016` ⇒ schreibe in Register `16.112`, lese von Register `16.121`
- ...

Über TINE sieht der Aufruf zum Auslesen der 5 bzw. 4 Datenwerte wie folgt aus:

- `/LINAC2/RF104-PIA.CDI/#2-#6` (5 Datenwerte), lesen der Endstufe mit der Property `RECV.CLBR` (Lesen und Calibrieren)
- `/LINAC2/RF104-PIA.CDI/#8-#11` (4 Datenwerte), lesen der Vorstufe mit der Property `RECV.CLBR` (Lesen und Calibrieren)

Die Applikation wird jeweils ein Datentyp `TDataType` für die Ausgabe (`output`) mit einem Array von 5 Daten und einer Eingabe (`input`) mit dem Datentyp `null` übergeben. Diese Parameter werden dem Konstruktor `TLink` mitgeteilt:

- `TLink link = new TLink("/LINAC2/RF104-PIA.CDI/#2-#6",RECV.CLBR, new TDataType(new float[5]), null, TAccess.CA_READ)`

Wichtig ist hier der Aufbau der Adresse. Hier wird im Anschluss an das Crate und die Subadresse über einen Doppelpunkte getrennt die zu lesende und schreiben Adresse als Offset übergeben. In unserem Fall also die Anzahl der zu lesenden Register.
CDI erkennt nun, dass wenn dieser Link über TINE abgearbeite wird, dass **KEINE** Eingabeparameter vorhanden sind. Also schaut es in der internen Tabelle nach und findet die `-22016`, welchen CDI in die Hardware schreibt. Anschließend wird der Wert gelesen und zurückgegeben.

3.7 Beispiele von CDI Tabellen

Als Beispiele seien hier die real existierenden CDI Aufbauten für HF unter TINE genannt. Zugegriffen wird auf diese Tabellen z.B. über den Java Instant Client (`\\mcalaunch.desy.de\webapps$\Released\common\Service\InstantClient.jnlp`).

- `/LINAC2/RF104-PIA.CDI/...`, Rechner: `ACCXPL2R2C`

- `/LINAC2/RF125-PIA.CDI/...`, Rechner: `ACCXPL2R2C`
- `/LINAC2/RF.Attenuator.CDI/...`, Rechner: `???`[33]
- `/LINAC2/RF.Modulator.CDI/...`, Rechner: `???`
- `/LINAC2/RF.Multiplexer.CDI/...`, Rechner: `???`
- `/LINAC2/RF.Phase.CDI/...`, Rechner: `???`
- `/LINAC2/RF.SLED.CDI/...`, Rechner: `???`
- `/LINAC2/RF.Various.CDI/...`, Rechner: `???`

Die einzelnen Tabellen befinden sich auf den Rechnersystemen unter dem Pfad `L:\server...` und den entsprechenden Unterverzeichnissen (z.B. für die PIA HF 10,4 MHz Sender unter `rf104`).

4 Systemaufbau und -funktionsweise

Der folgende Abschnitt behandelt die Entwicklung eines CDI Device-Servers in Eigenregie. Wer statt dessen den Weg über den DeviceServer von Josef Wilgen gehen möchte, der sollte sich für eine erste Dokumentation und Einführung an ihn wenden.

4.1 Minimale CDI Applikation

Der CDI-Server schafft die Verbindung zwischen der Hard- und der Software, indem er als Zwischenschicht die Anfragen für die Software auf die entsprechenden Register delegiert. Umgekehrt stellt CDI aber auch Register-Daten der Software über Properties zur Verfügung. Der Server muss von dem Entwickler selbst geschaffen werden. Es muss darauf geachtet werden, dass einmal die Klassenmethode `CDI.start()` durchlaufen werden muss. Diese Funktion ruft die einzelnen Konfigurationsdateien auf und initialisiert sich.[34]

Der folgende Beispiel-Server kann als Ausgang verwendet werden:

Beispiel-Server

```
1 package de.desy.mst....;
2
3 import de.desy.cdi.Cdi;
4
5 /**
6  * This is an example server for reading in the configuration files and initialize
7  * of CDI.
8  *
9  * @version 0.0.2
```

[33] **Ist noch auszufüllen!**

[34] Setzt einen Hook und initialisiert den Zugriff durch den Localhost über "`/localhost/cdi/...`". Dieses entspricht nicht den oben angeführten Remote-Zugriffen über z.B. "`/LINAC2/RF104-PIA.CDI`" oder "`/LINAC2/RF.Attenuator.CDI`" oder dergleichen. Der lokale Zugriff ist performanter als ein Remotezugriff, da hier nicht der Umweg über eine Netzwerkverbindung gegangen werden muss.

```
10  * @date 08.03.2007
11  */
12 public class CDIServer
13 {
14     /**
15      * This is the main class method which will initialize CDI.
16      *
17      * @param args_ the command line arguments.
18      */
19     public static void main(String[] args_)
20     {
21                 ...
22                 Cdi.start();
23                 ...
24     }
25 }
26 /* end of file */
```

Dieser Server wird über eine in der Tabelle `fecid.csv` verwendeten Device Context und Server angesprochen. Die einzelnen Devices werden über ihren Device Namen geholt, wobei dieser über bestimmte Eigenschaften oder Properties verfügt die eingestellt werden können.

4.2 Funktionsweise des CDI Servers

Der abstrakte Aufbau des CDI Servers wird in dem folgenden Abschnitt erklärt. Zunächst muss die oben ausgeführte CDI Applikation gebaut werden. Beim Starten dieser Applikation werden nun die folgenden Schritte für das Einladen der Konfigurationsdateien durchlaufen:

`MUSS`: Die Datei `fecid.csv` wird eingeladen und abgearbeitet.

`MUSS`: Die Datei `cdimf.csv` wird eingelesen und die unterschiedlichen Busse werden umgesetzt. Das heißt, es werden die entsprechenden DLL bzw. SO Bibliotheken in den Cache geladen. Sie werden unter den Namen des `BUS_ENV` der Initializierung über eine Tabelle zur Verfügung gestellt.[35]

`MUSS`: Die Datei `cdiaddr.csv` wird eingelesen und Zeilenweise abgearbeitet.

`KANN`: Die Datei `fecname.csv` wird eingeladen und die Zeile werden registriert.

`KANN`: Die Datei `export.csv` wird eingeladen und die Zeilen werden registriert.

`KANN`: Die Datei `history.csv` wird eingeladen und die Zeilen werden registriert.

[35] Siehe im Abschnitt 3.1.

Die CDI-Applikation schafft nach dem Einlesen der Dateien `cdimf.csv` und `cdiaddr.csv` zwei Server. Dieses sind der **Raw-Server**[36] und der **Device-Server**[37]. Der Raw-Server ermöglicht es über bestimmte, von CDI fest vorgegebene Properties Daten auszulesen, den Zustand des CDI Servers zu erfragen oder zu steuern. Der Raw-Servername ist der **FEC-ID** Namen der Datei `fecid.csv` oder der Computernamen[38], mit einem abschließendem `.CDI`. Fehlen die Konfigurationsdateien `fecname.csv` und `exports.csv` so wird ein leerer Device-Server geschaffen, der keinerlei Properties anbietet. Im Programmtext des Servers muss die Methode `Cdi.Start()` aufgerufen werden, am besten direkt in der `main` Klassenmethode (siehe oben).

Es existiert aber neben der Verwendung der `export.csv` auch die Möglichkeit die Properties im Programmtext explizit zu verwirklichen. Diese werden direkt über den Aufruf der Methode `de.desy.tine.server.equipment.TEquipmentModule.registerProperties()` und `de.desy.tine.server.equipment.TEquipmentModule.registerDevices()` registriert.

[36] Dieser Server wird von CDI ohne zusätzlichen Aufwand des Softwareentwicklers geschaffen und stellt die Properties bzw. Device Namen direkt dem Benutzer zur Verfügung. Über die Properties können bestimmte Unterfunktionalitäten der einzelnen Devices abgerufen werden. So z.B. die `UNITS` aus dem Anschnitt 3.2.2

[37] Dieser Server muss vom Softwareentwickler selbst entworfen werden. Er liest die Device Namen direkt auf dem lokalen Rechner über den Device Context `/localhost/cdi` ein. Diese Daten im Server verarbeitet und über selbst definierte Properties zur Verfügung gestellt.

[38] Falls es keine `fecid.csv` gibt.