GINA2010

Benutzerdokumentation

rd electronic GmbH
Zweigstelle Dresden
Bernhardstraße 70
01187 Dresden

Tel. +49 351-6563-56-4
Fax +49 351-6563-56-5

Internet: http://www.rd-electronic.de

Dokumentnummer	:	GINA2010-UserMan
Ausgabe	:	2.0.0.2
Status	:	Released
Letzte Änderung	:	30.03.2021
Erstellung	:	26.04.2018
Autoren	:	MBa, JWo, SBa

Rechtliche Hinweise

Änderungsübersicht

Version	Datum	Autor	Änderung
2.0.0.0	26.04.2018	SBa	• Neuerstellung für PDF
2.0.0.1	02.05.2018	SBa	• Verfeinerung von Formulierungen
2.0.0.2	30.03.2021	SBa	• Überarbeitung, Release

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Allgemeines

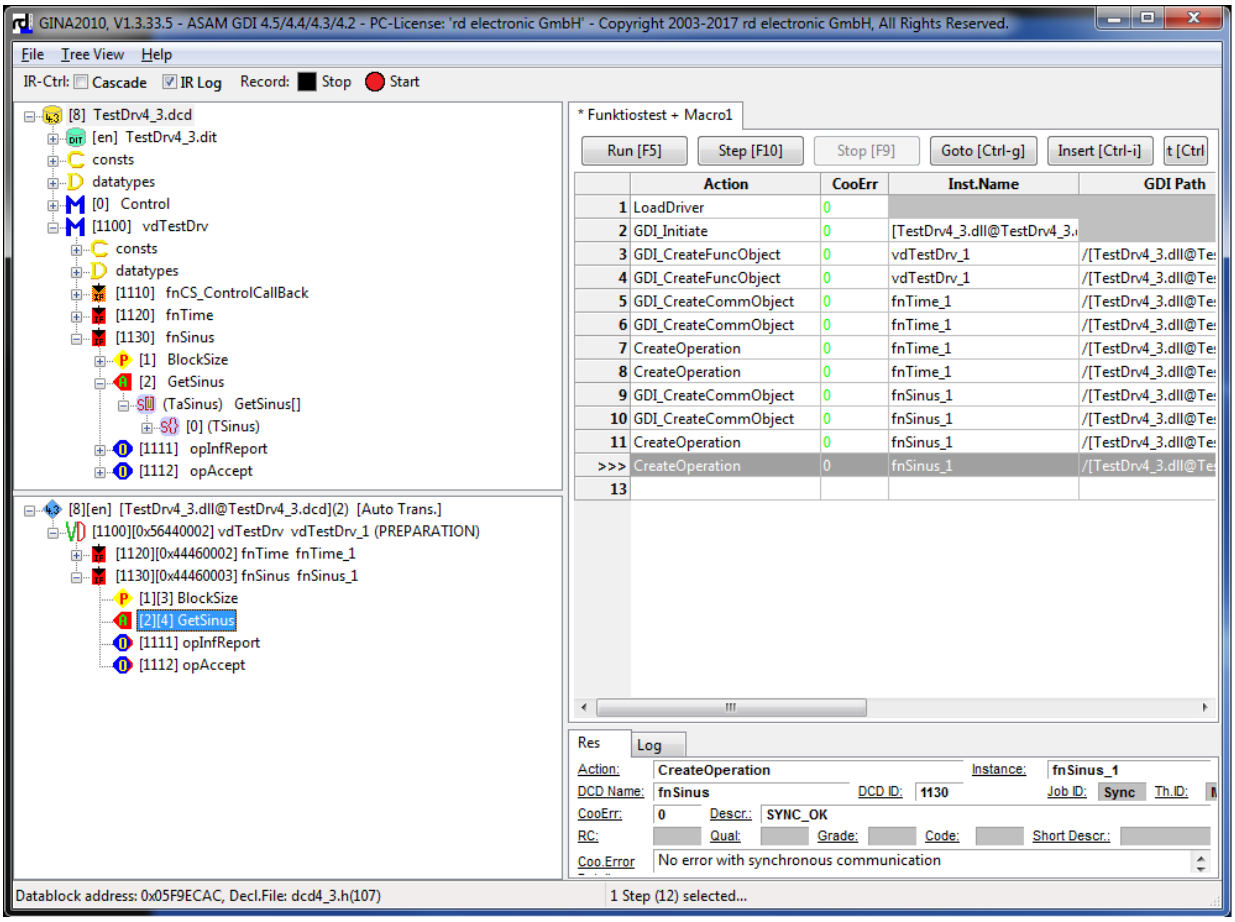


Bild 1: GINA2010 – Allgemein

1.1 Identifikation

Dieses Dokument beschreibt die Bedienung der Anwendung **GINA2010** der Firma rd electronic GmbH (**RDE**) und des integrierten **Skeleton Generators** zur Generierung von **Device Driver Skeletons** (**ASAM GDI** Gerätetreiber-Gerüsten) für den Ausbau zu vollwertigen **Device Drivers** (**ASAM GDI** Gerätetreibern).

1.2 Ziel

Das Dokument soll in erster Linie **Anlagenlieferanten** Hilfestellung beim Test und bei Vorabnahmen der ihren Anlagen zugehörigen **Device Drivers** mit **GINA2010** geben.

Weiterhin sollen **Anlagenlieferanten** in die Lage versetzt werden, mittels des integrierten **Skeleton Generator** eigene **Device Driver Skeletons** zu erstellen, um diese zu vollwertigen **Device Drivers** entsprechend ihren Anlage bzw. Geräten auszubauen.

1.3 GINA2010-Dokumente

[RDE-Doc1] **"GINA2010 - Benutzerdokumentation"**
Datei : GINA2010-UserMan.pdf
Version : 2.0.0.2
Datum : 30.03.2021

[RDE-Doc2] **"Abspeicherung von IR Daten"**
Datei : GINA2010-UserMan-IrData.pdf

[RDE-Doc3] **"GINA2010 - Reporterstellung"**
Datei : GINA2010-UserMan-Report.pdf
Version : 1.1.0.0
Datum : 03.12.2014

[RDE-Doc4] **"Device Driver Skeleton - Benutzerdokumentation "**
Datei : GINA2010-UserMan-Skeleton.pdf
Version : 0.0.0.0
Datum : 19.04.2018

[RDE-Doc5] **"GINA2010 - Makrosprache"**
Datei : GINA2010-UserMan-Macro.pdf
Version : 1.2.0.0
Datum : 05.03.2014

1.4 Referenzierte Dokumente

[GDI-Doc1] **"Specification of the Device Capability Description of an ASAM-Device Driver"**

Datei : Gdi-Part_A_Ver420_en.pdf
Titel : ASAM-GDI Part A
Version : 4.20
Datum : 01.02.2001

[GDI-Doc2] **"Specification of the Interface of an ASAM-Device Driver"**

Datei : Gdi-Part_B_Ver420_en.pdf
Titel : ASAM-GDI Part B
Version : 4.2
Datum : 02.02.2001

[GDI-Doc3] **"Generic Device Interface Specification - Version 4.3.2 - Specification"**

Datei : ASAM_GDI_4_3_2_Specification.pdf
Version : 4.3.2 (Maintenance Release)
Datum : 15.02.2005

[GDI-Doc4] **"ASAM GDI - Version 4.4.0 - Specification"**

Datei : ASAM_GDI_V4.4.0_Specification.pdf
Version : 4.4.0 (Release)
Datum : 31.01.2008

[GDI-Doc5] **"ASAM GDI - Generic Device Interface - Part 1 of 3"**

Datei : ASAM_CAT_GDI_BS_1_3_Programmers-Guide-Base_V4-5-0.pdf
Version : 4.5.0
Datum : 31.01.2011

[GDI-Doc6] **"ASAM GDI - Transport Layer Communication Type COM"**

Datei : ASAM_GDI_Transport-Layer-Communication-Types-COM_V3-0-0.pdf
Version : 3.0.0
Datum : 08.08.2008

[GDI-Doc7] **"ASAM GDI - Transport Layer Communication Type IP4"**

Datei : ASAM_GDI_Transport-Layer-Communication-Types-IP4_V3-0-0.pdf

Version : 3.0.0

Datum : 08.08.2008

1.5 Definitionen und Abkürzungen

Abstract Interface	Nicht instanziierbares vererbbares **Interface** [GDI-Doc3]
Accept	Callback-Mechanismus auf **Communication Objects** zur asynchronen, durch einen **Device Driver** angeforderten, Übertragung von Daten aus der Anwendung an den **Device Driver**
Anlagenlieferanten	Lieferant eines Gerätes oder einer Anlage
API	Application Programmers Interface, Programmierschnittstelle einer Softwarekomponente
ASAM	Association of Standardisation of Automation and Measuring Systems
Attribute	**DCD**-Typ für **Communication Objects** zum Datenaustausch in **Function Objects**
Betriebsmodus	Der Modus enthält die zwei Ausprägungen „Automatisch“ zur automatischen Bedienung der **Transition Functions** durch die **Coordinator Engine** und „Manuell“ zur Bedienung der **Transistion Functions** durch den Benutzer
Boolean	**ASAM GDI** Datentyp [GDI-Doc3]
Char	**ASAM GDI** Datentyp [GDI-Doc3]
Create Parameter	Spezieller **Parameter** (**Communication Object**) zum einmaligen Parametrieren von **Function Objects**
Communication Object	**DCD**-Typ bzw. Instanz eines solchen nach **ASAM GDI** zum Austausch von Informationen zwischen Anwendung und **Device Driver** [GDI-Doc3] Auch als **Attribute** bezeichnet
Communication Type	Transport Layer Communication Type (COM [GDI-Doc6], IP4 [GDI-Doc7])
Companion	Spezifikation eines Geräteprofils nach **ASAM GDI** [GDI-Doc3]
Coordinator	**ASAM GDI Coordinator** [GDI-Doc3] der **RDE** mit spezieller Anwendungsschnittstelle für die Volkswagen AG
Coordinator Engine	In **GINA2010** integrierte Kernkomponente zur Ansteuerung von **Device Drivers** nach dem **ASAM GDI Standard** Die **Coordinator Engine** findet auch im **Coordinator** der **RDE** Einsatz.

Control VD	**Virtual Device** zur Steuerung der **GDI Phasen** [GDI-Doc3]
Double	**ASAM GDI** Datentyp [GDI-Doc3]
DCD	Device Capability Description, Datei, die die Fähigkeiten eines Gerätes abstrahiert beschreibt. Enthält Informationen über die instanziierbaren Objekte und definierte Datentypen sowie Konstanten [GDI-Doc3]
Device Base Function	**Function Object** des **Control VD** mit optionalen **Operationen**
Device Driver	**ASAM GDI** Gerätetreiber nach dem Standard **ASAM GDI** [GDI-Doc3]
Device Driver Skeleton	Mittels **Skeleton Generator** generiertes Gerüst zur Erstellung eines **Device Driver**
Device Function	Alternative Bezeichnung für **Interface**
DIT	Device Info Text, Datei, die verschiedene, beschreibende Strings enthält. Sie ist eine Ergänzung zur **DCD** und kann in verschiedenen Sprachen vorliegen [GDI-Doc3]
DLL	Dynamic Linked Library, Bezeichnung einer dynamischen Bibliothek unter Windows mit der Dateiendung *.dll (analog **DSO** unter Linux)
DSO	Dynamic Shared Object, Bezeichnung einer dynamischen Bibliothek unter Linux/Unix mit der Dateiendung *.so (analog **DLL** unter Windows)
Enumeration	**ASAM GDI** Datentyp [GDI-Doc3]
Enumerator	Spezielle Ausprägung innerhalb einer Enumeration [GDI-Doc3]
Float	**ASAM GDI** Datentyp [GDI-Doc3]
Function Reference	**DCD-Typ** zur Referenzierung eines **Function Object** [GDI-Doc3]
Function Object	Instanz des **DCD**-Typs **Interface** [GDI-Doc3]
GDI	Generic Device Interface, **ASAM GDI** Standard [GDI-Doc1], [GDI-Doc2], [GDI-Doc3], [GDI-Doc4], [GDI-Doc5]
GDI Phase	Zustand nach **ASAM GDI** Phasenmodell [GDI-Doc3]
Information Report	Callback-Mechanismus auf **Communication Objects** zur asynchronen Übertragung von Daten vom **Device Driver** zur Anwendung
Initiate Parameter	Spezieller **Parameter** (**Communication Object**) zum initialen Parametrieren von **Function Objects**
Interface	**DCD**-Typ nach **ASAM GDI** als Container für **Communication Objects** und **Operations** [GDI-Doc3] Auch als **Device Function** bezeichnet
LicManClient	Lizenzierungstool der **RDE** zur Aktivierung von Einzelplatzlizenzen

Limited Sequence	In ihrer Länge begrenzte **Sequence** [GDI-Doc3]
Long	**ASAM GDI** Datentyp [GDI-Doc3]
Makro	In der Schrittliste [5.7] aufgeführte Ausführungsschritte
Module	**DCD**-Typ nach **ASAM GDI** als Container für **Interfaces** [GDI-Doc3]
Octet	**ASAM GDI** Datentyp [GDI-Doc3]
Operation	**DCD**-Typ nach **ASAM GDI** zum Ausführen von Aktionen an Anlagen und Geräten
PA	**Platform Adapter**
Parameter	**DCD**-Typ für **Communication Objects** zur Parametrierung von **Function Objects**
Platform Adapter	**ASAM GDI** Bibliothek zur Nutzung für plattformunabhängige **Device Driver**. Besitzt eine standardisierte Schnittstelle und abstrahiert systemspezifische Funktionalitäten [GDI-Doc3]
Platform Adapter Extension	**ASAM GDI** Platform Adapter Extension [GDI-Doc3]
RDE	Firma rd electronic GmbH
Selector	Bestandteil des **ASAM GDI** Datentyps **Union** [GDI-Doc3]
Sequence	**ASAM GDI** Datentyp [GDI-Doc3]
Short	**ASAM GDI** Datentyp [GDI-Doc3]
Skeleton Generator	In **GINA2010** integrierte Komponente zur Erstellung von **Device Driver Skeletons** nach dem **ASAM GDI** Standard (zusätzliche Lizenz erforderlich)
Standard Extension	**Platform Adapter Extension** mit **Communication Types** for Ethernet und serielle Verbindungen der Firma RDE
Struktur	**ASAM GDI** Datentyp [GDI-Doc3]
Transition	Umschaltung in eine andere **GDI Phase**
Transition Function	**Function Object** des **Control VD** mit **Operationen** zur Schaltung der **GDI Phasen**
ULong	**ASAM GDI** Datentyp [GDI-Doc3]
Union	**ASAM GDI** Datentyp [GDI-Doc3]
Unlimited Sequence	In ihrer Länge unbegrenzte **Sequence** [GDI-Doc3]
UShort	**ASAM GDI** Datentyp [GDI-Doc3]
VD	**Virtual Device**
Virtual Device	Instanz des **DCD**-Typs **Module**. Repräsentiert einen bestimmten Aspekt eines Gerätes, das über einen **Device Driver** angesprochen wird [GDI-Doc3]

2 Einleitung

2.1 Was ist GINA2010

GINA2010 ist ein in C++ geschriebenes Tool zur Ansteuerung von **Device Drivers** für alle Arten von Anlagen und Geräten. Es ist zur Entwicklung und zum Test von **Device Drivers** konzipiert. Dabei ist es möglich die einzelnen Funktionen eines **Device Driver** über die eingebaute **Coordinator Engine** aufzurufen. Sequenzen dieser Funktionsaufrufe können in **Makro**-Dateien abgelegt und später erneut ausgeführt werden. Grundlegend dazu unterstützt **GINA2010** das Einlesen von **DCD**-Dateien und die grafische Darstellung dieser Inhalte.

Um der plattformunabhängigen Konzeption von **GDI** Rechnung zu tragen, ist es möglich, **GINA2010** unter verschiedenen Betriebssystemen zu benutzen. Um dies zu erreichen, wurde das plattformübergreifende grafische Toolkit „wxWidgets“ verwendet.

GINA2010 ist für die **GDI** Versionen 4.2, 4.3, 4.4 und 4.5 konzipiert.

2.2 Beschaffung und Lizenzen

Nutzungsrechte an **GINA2010** und dem integrierten **Skeleton Generator** können in verschiedenen Lizenzformen erworben werden.

Üblicherweise werden hier Einzelplatz-Lizenzen angeboten, welche an spezielle Rechner gebunden und zur Kostenreduktion bedarfsgerecht zeitlich beschränkt werden.

Umfassende Informationen zu Lizenzen und Preislisten erhalten Sie auf eine Anfrage an license.support@rd-electronic.de.

2.3 ASAM-GDI Vorkenntnisse

Es ist zweckmäßig sich im Vorfeld der Benutzung mit den Konzepten und Möglichkeiten von **ASAM GDI** vertraut zu machen. Als Referenz sollte die Spezifikation zu **ASAM GDI** 4.3.2 [GDI-Doc3] herangezogen werden. Für ältere **Device Driver** kann es auch sinnvoll sein, die **ASAM GDI** 4.2 Spezifikationen [GDI-Doc1] und [GDI-Doc2], für neuere **Device Driver** die Spezifikationen zu **ASAM GDI** 4.4 [GDI-Doc4] und zu **ASAM GDI** 4.5 [GDI-Doc5] heranzuziehen.

Weiterhin sollten Kenntnisse in der Funktionsweise des verwendeten Betriebssystems vorliegen. Das betrifft insbesondere die Handhabung von dynamischen Bibliotheken (**DLL**, **DSO** o.ä.) und der Prozessumgebung (Umgebungsvariablen).

Kenntnisse in der für **Device Driver** vorgeschriebenen Programmiersprache ANSI C/C++ sind für die Bedienung des Tools nicht erforderlich, können jedoch hilfreich sein.

3 Installation

Die verwendete Installationsprozedur ist von der Zielplattform abhängig. Im Folgenden sind die Installationsprozeduren für die üblichen Plattformen erläutert.

3.1 Windows

Die Windows Installation kann mit Hilfe eines InstallShield-Setups durchgeführt werden. Die Installation kann deshalb auch durch Kopieren in ein Verzeichnis eigener Wahl erfolgen.

3.1.1 Voraussetzungen

GINA2010 ist derzeit unter den Windows-Systemen NT4, 2000, XP, 2003, 7, 8 und 10 getestet. Ältere NT Systeme sowie Windows 95, 98, ME werden nicht unterstützt.

3.1.2 Ablauf

Zur Installation ist das Programm `GINA2010_Setup.exe` auszuführen. Die eingeblendeten Dialoge und Meldungen sind in üblicher Weise auszufüllen bzw. zu bestätigen. Unter anderem kann auch das Installationsverzeichnis ausgewählt werden.

Im letzten Dialog kann die Installation durch Anklicken des Schalters Install ausgelöst oder des Schalters Cancel abgebrochen werden.

Während des Installationsprozesses werden folgende Schritte durchgeführt:

- Erstellung eines Eintrags im Start-Menü `"rd electronic gmbh\GINA2010"` mit Links zum **GINA2010**-Programm, zum Deinstallationsprogramm, zu **GINA2010**-Hilfedateien und zum Lizenzierungsprogramm **LicManClient** zur Registrierung von Einzelplatzlizenzen.

- Registrierung der Dateiendung `*.gam` für **GINA2010**. Die durch **GINA2010** erstellten **Makro**-Dateien werden mit dieser Dateiendung versehen.

- Erzeugung der folgenden Dateien und Verzeichnisse unterhalb des Installationsverzeichnisses:

  - `GINA2010.exe`

    Monolithische Release Version des Tools (Enthält **DCD-Parser**, **DIT-Parser**, **Coordinator Engine** und wxWidgets Bibliotheken).

  - `LicManClient.exe`

    Ist eine Lizenzierungsanwendung zum Bezug von Einzelplatzlizenzen für **GINA2010**.

  - `PAWINNT4_5.dll, PAWINNT4_4.dll, PAWINNT4_3.dll, PAWINNT.dll`

    Platform Adapter nach **GDI** 4.5 mit Wrapper-Dateien auf ältere **GDI**-Versionen

  - `rde_paext_std_433.dll, STDPAEXT4_3.dll`

    Standard Extensions mit den **Communication Typs** für Ethernet- und serielle Gerätekommunikation.

    Um sicher zu gehen, dass die gewünschten **Platform Adapter Extensions** gefunden werden, sollte zusätzlich die Umgebungsvariable `GDI_PAEXT_DIR` [3.5] gesetzt werden. Sie sollte auf das Installationsverzeichnis zeigen (Default: „`C:\Programme\rd electronic Gmbh\GINA2010`“) zeigen.

  - `rde_pa_config.cfg, rde_paext_conn_names.cfg`

    Konfigurationsdateien zum Routen von verschiedenen **Communication Types** zu den entsprechenden **Extensions** und zum Mappen von **ASAM GDI** Standard Namen auf entsprechende systemabhängige Verbindungsnamen (e.g. `COM1` → “`/dev/ttyS0`“ für Linux-Systeme)

  - `APIHeader/`

    Das Verzeichnis enthält die Headerdateien des **ASAM GDI** Standards für **Platform Adapter**, **Extensions** und **Device Driver**.

- `BaseDITs/`

  Das Verzeichnis enthält ASAM GDI Standard DIT Dateien zur weiteren Verwendung für **Device Driver**.

- `DCDHeader/`

  Verzeichnis, enthält die benötigten **Com_Error DITs** und **DCD-Header** für **GDI** 4.2, **GDI** 4.3.2., **GDI** 4.4 und **GDI** 4.5. Diese sind gegebenenfalls in das Verzeichnis der zu verwendenden **DCD** zu kopieren, wenn sie dort nicht bereits vorhanden sind.

- `Documentation/`

  Das Verzeichnis enthält die Dokumentationen zum **GINA2010**.

- `PALibs/`

  Das Verzeichnis enthält die Importbibliotheken des **Platform Adapters** zur Importierung in Windows-Projekte.

- `PALog/`

  Verzeichnis, zur Ablage der vom **Platform Adapter** generierten Log-Dateien.

  Das Verzeichnis ist in der entsprechenden Umgebungsvariable `GDI_PALOG_DIR` [3.5] aufzunehmen.

- `XMLReport/`

  Verzeichnis, zur Ablage der vom **GINA2010** generierten Report-Dateien.

3.1.3 Platform Adapter

Der mitgelieferte **Platform Adapter** ist entsprechend der erworbenen Lizenz [2.2] unter Umständen nicht in anderen Bereichen einsetzbar.

Entsprechend **ASAM GDI** ist es jedoch möglich, **Platform Adapters** anderer Hersteller in **GINA2010** zu verwenden, vorausgesetzt sie haben den gleichen Namen und ein Alignment von `8`. Dieses ist für Windows Versionen der **RDE Platform Adapters** festgelegt. Außerdem sind die folgenden Aufrufkonventionen zu beachten.

Die exportierten Funktionsnamen müssen mit „`_Name@n`" dekoriert sein, wobei `Name` der Funktionsname und `n` die dezimale Anzahl der Bytes in der Parameterliste ist. Dies entspricht der verwendeten Windows Aufrufkonvention `__stdcall`. Diese unterscheidet sich auch hinsichtlich des Stack-Handling vom C-Standardaufruf `__cdecl`. Der Stack wird dabei von der aufgerufenen Funktion bereinigt.

Eine Ausnahme bilden hierbei die CallBack-Funktionen `io_complete` und `io_event`. Diese müssen mit der Aufrufkonvention `__cdecl` deklariert werden. Der Stack wird bei dieser Deklaration von der aufrufenden Funktion wieder bereinigt. Der dabei theoretisch definierte Präfix des Funktionsnamens „`_`" spielt hier keine Rolle weil die Funktionen über übergebene Zeiger aufgerufen werden und nicht über Exporte.

Beide Deklarationen der Aufrufkonvention, `__stdcall` und `__cdecl`, sind Windows-spezifisch und können nicht auf andere Systeme übertragen werden.

Es ist nicht möglich **GINA2010** ohne **Platform Adapter** zu starten.

3.1.4 Platform Adapter Extensions

Der **Platform Adapter** dekoriert unter Windows den angeforderten Namen der **Platform Adapter Extension** mit der Dateierweiterung „`.dll`", um den Dateinamen zu ermitteln

- `[Extension Name].dll`

z.B. würde die **Platform Adapter Extension** bei der Angabe von „`Test`" die Bibliothek `Test.dll` suchen [siehe 3.5].

Die Aufrufkonventionen der **Platform Adapter Extension** entsprechen denen des **Platform Adapter** [3.1.3].

3.1.5 Registry

Fensterpositionen und andere Konfigurationsinformationen werden in der Registry abgelegt, wenn die Sicherung der Konfiguration [5.3.5] durch den Benutzer durchgeführt wird.

Nach Erwerb einer Einzelplatzlizenz [2.2] werden durch das mitgelieferte **RDE** Lizenzierungstool **LicManClient** weitere Einträge vorgenommen, welche zur Laufzeit von **GINA2010** ausgewertet werden.

3.2 Linux

Die Linux Version wird als Tar-Archiv (Dateierweiterung `.tgz`) geliefert. Dieses Archiv enthält alle nötigen Dateien. Was bei der Installation zu beachten ist, wird im Folgenden erklärt.

3.2.1 Voraussetzungen

Durch die weit größere Vielfalt der Möglichkeiten, die im Rahmen von Linux Distributionen zur Verfügung stehen, gibt es grundsätzlich verschiedene Varianten, ein angestrebtes Ziel zu erreichen. Das verwendete Toolkit wxWidgets kann auf verschiedene, unter Linux Distributionen üblichen Toolkits (X11 native, GTK1.2, GTK2.0, Motiv) aufsetzen.

Die aktuelle Version setzt auf GTK1.2 auf. Damit wurden die besten Ergebnisse bezüglich möglichst identischen Verhaltens zwischen verschiedenen Betriebssystemen erzielt. Außerdem ist eine kompatible Version bereits unter Suse8.2 (oder früher) verfügbar.

Die Entwicklung der Linux Version ist unter der Suse 10.0 Distribution erfolgt. Dabei wurde der Compiler GCC 4.02 verwendet. Im Folgenden werden die Abhängigkeiten beschrieben.

- Folgende Bibliotheken (Shared Objects, **DSOs**) werden benötigt. Es ist dabei zu beachten, dass die hier aufgeführten Namen die Shared-Object-Namen (soname) sind. Diese sind nicht zwangsläufig als Dateien im System vorhanden, sollten aber als symbolischer Link eingerichtet sein (erzeugt normalerweise `ldconfig`).

libgtk-1.2.so.0	libpthread.so.0	libXxf86vm.so.1	libm.so.6
libgdk-1.2.so.0	libXi.so.6	libpng.so.3	libpalinux4_3.so
libgmodule-1.2.so.0	libXext.so.6	libexpat.so.0	libstdc++.so.6

libgthread-1.2.so.0	libX11.so.6	libz.so.1	libgcc_s.so.1
libglib-1.2.so.0	libXinerama.so.1	libdl.so.2	libc.so.6

- Es wird eine GTK+ 1.2 Installation mindestens der Version 1.2.3 benötigt.
- Es wurde ein 2.6er Kernel bei der Entwicklung verwendet. Ein anderer Kernel sollte allerdings keine Auswirkungen haben.

Es sollte nicht zwingend notwendig sein, die zu testenden **Device Driver** oder **Platform Adapter Extensions** ebenfalls mit einem GCC4.x Compiler zu kompilieren. Solange keine dekorierten Symbolnamen exportiert werden (C++ Klassen, Namespaces, ...) und kein Exception Handling verwendet wird, sollten keine Probleme auftreten. Beides ist bei den ANSI C APIs von **Device Driver**, **Platform Adapter** und **Platform Adapter Extensions** nicht der Fall.

3.2.2 Andere Versionen

Es ist möglich, auch Versionen für andere Toolkits zu erzeugen. Genauso ist es möglich, Versionen zu erzeugen, die mit anderen Systembibliotheken gelinkt wurden oder mit anderen Compilerversionen kompiliert wurden (sollte dies notwendig sein).

Wenden Sie sich dazu bitte an den **RDE GDI** Support über GDI-Support@rd-electronic.de.

3.2.3 Installationshinweise

- Die Binärdatei des Programms muss in keinen speziellen Pfad kopiert werden. Die Dateiattribute müssen jedoch die Datei als für den aktuellen Nutzer ausführbar kennzeichnen.

- Der beiliegende **Platform Adapter** `libpalinux4_5.so.7.0.4.0` (derzeit und am Beispiel in Version 7.0.4.0) unterstützt die **GDI**-Versionen 4.2, 4.3, 4.4 und 4.5 und sollte zusätzlich durch folgende symbolische Links referenziert werden.
  - `libpalinux4_3.so`
  - `libpalinux4_3.so.5`
  - `libpalinux4_4.so.6`
  - `libpalinux4_5.so.7`

    Diese Links entsprechen der Konvention für SO Namen (soname) unter Linux und werden normalerweise von `ldconfig` automatisch erzeugt. Wird `ldconfig` nicht verwendet oder hat der **Platform Adapter** die SO-Namen (soname) nicht eingetragen, sollten diese Links trotzdem erzeugt werden. Möglicherweise sind Applikationen oder andere **DSOs** dagegen gelinkt.
  - `libpalinux.so`
  - `libpalinux4_3.so`
  - `libpalinux4_4.so`
  - `libpalinux4_5.so`

    Gegen diesen Namen linken **Device Drivers** normalerweise in Abhängigkeit der **GDI**-Version. Wurde gegen einen **Platform Adapter** ohne SO-Namen (soname) gelinkt, so wird auch dieser Link zur Laufzeit benötigt. Anderenfalls wird der oben beschriebene Link gesucht.

- Unter Linux ist es üblich, allgemeingültige Laufzeit Bibliotheken (**DSOs**) im System zu registrieren. Dies erfolgt durch Eintragung des die Laufzeit Bibliotheken enthaltenden Verzeichnisses in die Datei `/etc/ld.so.conf` und anschließendem Aufruf des Befehls `ldconfig`. Ist dies geschehen werden alle registrierten Komponenten (**Platform Adapter**, **Platform Adapter Extensions**, **Device Driver**, ...) von jeder Verzeichnisposition aus gefunden, solange beim linken oder expliziten Laden kein Pfad angegeben wird. Dieser Weg ist zu empfehlen, aber nicht zwingend notwendig.

- Sind die Laufzeit Bibliotheken (**Platform Adapter**, **Platform Adapter Extensions**, **Device Driver**, ...) nicht im System registriert, so müssen sie mit komplettem Pfad gesucht werden. Dies ist beim **Platform Adapter** nicht möglich. Für **Device Driver** und **Platform Adapter Extensions** [3.3] ist dies möglich. Damit auch der **Platform Adapter** gefunden werden kann muss die Umgebungsvariable `LD_LIBRARY_PATH` auch auf das den **Platform Adapter** enthaltende Verzeichnis zeigen. Dies gilt auch, wenn der **Platform Adapter** in demselben Verzeichnis wie die Binärdatei des Programms liegt.

3.2.4 Verzeichnisstruktur

Innerhalb der **RDE** hat sich unter Linux eine einheitliche Verzeichnisstruktur durchgesetzt, die ich hier kurz vorstellen möchte. Sie ist den unter Unix üblichen Bezeichnungen entlehnt. Es ist nicht zwingend notwendig dieses System zu benutzen.

Alle **GDI** Komponenten sind unterhalb eines Verzeichnisses abgelegt, welches in der Umgebungsvariable `GDI_HOME` eingestellt ist. Derzeit ist diese auf `/opt/GDI/` eingestellt.

Darunter befinden sich folgende Verzeichnisse:

- `doc`

  Ablage der Dokumentationen der verwendeten Komponenten.

- `bin`

  Ausführbare Programme (z.B. **GINA2010**) und Shell Scripts.

- `lib`

  Alle Bibliotheken liegen hier. Dieses Verzeichnis ist auch in `/etc/ld.so.conf` eingetragen. Hier liegen auch alle **Platform Adapter Extensions** [3.3].

- `src`

  Ablage der Quellen, speziell der **Device Driver**.

- `var`

  Zielverzeichnis für Log Dateien, speziell des **Platform Adapters** [3.3].

- `etc`

  Eventuell nötige Konfigurationsdateien für das **GDI** System. Die Konfiguration von **GINA2010** zählt dabei allerdings nicht dazu.

3.2.5 Platform Adapter

Der mitgelieferte **Platform Adapter** ist entsprechend der erworbenen Lizenz [2.2] unter Umständen nicht in anderen Bereichen einsetzbar.

Entsprechend **ASAM GDI** ist es jedoch möglich, **Platform Adapters** anderer Hersteller in **GINA2010** zu verwenden, vorausgesetzt sie exportieren undekorierte Funktionsnamen und haben ein passendes Alignment. Eventuell muss ein passender symbolischer Link erzeugt werden [3.2.3]. Gleiches gilt für die **Platform Adapter Extensions**.

Es ist nicht möglich **GINA2010** ohne **Platform Adapter** zu starten.

3.2.6 Platform Adapter Extensions

Der **Platform Adapter** dekoriert unter Linux den Namen einer angeforderten **Platform Adapter Extension** auf bestimmte Weise, um den Namen des zu ladenden **DSO** zu ermitteln:

- `lib[Extension Name].so`

z.B. würde die **Platform Adapter Extension** „`Test`“ als `libTest.so` gesucht.

Ungeachtet dessen sollten **Platform Adapter Extensions** auch mit den systemtypischen Links [3.2.3] versehen werden, denn die eigentliche **DSO**-Datei enthält normalerweise die Versionsnummer in ihrem Dateinamen. Die mitgelieferten **Standard Extensions** heißen `librde_paext_std_433.so` und `libstdpaext4_3.so`.

Es ist zu beachten, dass unter Unix-Systemen die Groß-/Kleinschreibung unterschieden wird.

Zum Auffinden der Extensions zur Laufzeit ist entsprechend die Umgebung einzustellen [3.5.1.2].

3.2.7 Konfiguration

Die Fensterpositionen und andere Konfigurationsinformationen werden unter Linux in einer Konfigurationsdatei abgelegt, wenn die Sicherung der Konfiguration angefordert wird.

Das Format der Datei ähnelt den von KDE und Windows bekannten Konfigurationsdateien. Sie wird im Home Verzeichnis des aktuellen Benutzers abgelegt und beginnt (wie unter Unix üblich) mit einem Punkt, gefolgt vom Namen des Programms. Der Punkt kennzeichnet die Datei als versteckt.

Jeder Benutzer hat somit seine eigene Konfigurationsdatei. Es ist aber auch mögliche eine Default-Datei gleichen Namens unter `/etc/` abzulegen. Diese wird dann benutzt, wenn im Verzeichnis des Benutzers keine entsprechende Datei gefunden wurde bzw. gesuchte Einträge dort nicht vorhanden sind. Dies kann dazu dienen, bestimmte Informationen systemweit vorzukonfigurieren.

3.3 Andere Betriebssysteme

Prinzipiell ist die Verwendung von **GINA2010** auch unter anderen Betriebssystemen wie z.B. Mac OS oder FreeBSD möglich. Voraussetzung ist die Verfügbarkeit einer kompatiblen Portierung von wxWidgets für das gewünschte System und die Verfügbarkeit des für **GDI** ohnehin erforderlichen ANSI C/C++ Compilers.

Bislang sind Portierungen jedoch nur nach Windows und Linux erfolgt. Sollte Interesse an einer Portierung auf ein spezielles System bestehen wenden Sie sich bitte an unseren **GDI** Support unter der Email-Adresse GDI-Support@rd-electronic.de.

3.4 Alignment

Bei Erweiterung der **GINA2010**-Installation um weitere **Platform Adapter Extension** sind unbedingt die in den Kapiteln 4.1.1 und 4.1.2 getroffenen Alignment-Festlegungen einzuhalten.

3.5 Umgebungsvariablen

Generell können Umgebungsvariablen in einem Startskript oder im System gesetzt werden.

Es ist dabei zu beachten, dass Umgebungsvariablen unter Windows je nach Ort der Eintragung zu unterschiedlichen Zeitpunkten gültig werden. So kann z.B. der Neustart des entsprechenden Programms oder gar des ganzen Betriebssystems notwendig werden.

GINA2010 bietet zusätzlich eine grafische Benutzerschnittstelle [0] zur Verwaltung der in **RDE**-Softwarekomponenten verwendeten Umgebungsvariablen, aber auch weiterer benutzerdefinierter Umgebungsvariablen im Prozessraum.

3.5.1 Konfiguration von Verzeichnissen

3.5.1.1 GDI_PALOG_DIR (Logdatei-Verzeichnis)

Diese Umgebungsvariable steuert grundlegend das Logging über den **Platform Adapter**. Eine detaillierte Beschreibung ist unter Kapitel 3.5.2.1 zu finden.

3.5.1.2 GDI_PAEXT_DIR (Extension-Pfad)

Ist diese Umgebungsvariable verfügbar, so werden alle **Platform Adapter Extensions** in dem hier angegebenen Pfad gesucht.

- Ist der angegebene Pfad ein relativer Pfad, so richtet sich dieser am aktuellen Ausführungsverzeichnis von **GINA2010** aus.
- Ist diese Umgebungsvariable leer oder nicht gesetzt, so wird in den systemtypischen Verzeichnissen nach **Platform Adapter Extensions** gesucht. Dies wären bei Windows vor allem `System32/` und das aktuelle Verzeichnis (weitere in systemabhängiger Ordnung) und bei Linux die über `etc/ld.so.conf` registrierten Verzeichnisse und die Verzeichnisse, auf die die Umgebungsvariable `LD_LIBRARY_PATH` zeigt. Eventuelle abweichende „soname“-Einträge in den **DSOs** sind dabei zu beachten.

Diese Umgebungsvariable wird bei jedem Aufruf der **Platform Adapter**-Funktion `io_initiate` ausgelesen.

3.5.1.3 GDI_PA_CONF_DIR (Konfigurationsdatei-Pfad)

Diese Umgebungsvariable wurde ab **GDI** Version 4.3.3 eingeführt und beinhaltet den Verzeichnispfad der Konfigurationsdatei mit Einträgen, die die Zuordnung der von **Device Drivers** verwendeten **Communication Types** ([GDI-Doc6], [GDI-Doc7]) zu den zugehörigen **Platform Adapter Extensions** bei Aufrufen von `io_initiate` herstellen.

- Ist ein **Platform Adapter** ab Version 4.4 eingespielt (`PAWINNT4_4*.*`, `PAWINNT4_5*.*`, ...) wird die Routing-Datei mit dem Namen `rde_pa_config.cfg` im angegebenen Verzeichnis gesucht.
- Vorgängerversionen des **Platform Adapter** suchen die Routing-Datei mit dem Namen `rde_pa_ct_routing.cfg` im angegebenen Verzeichnis.
- Ist der in dieser Umgebungsvariablen eingetragene Verzeichnispfad ein absoluter Pfad, so wird dieser verwendet.
- Ist der in dieser Umgebungsvariablen eingetragene Verzeichnispfad ein relativer Pfad, so richtet sich dieser am aktuellen Ausführungsverzeichnis von **GINA2010** aus.
- Ist die Umgebungsvariable leer, nicht gesetzt oder mit einem Verweis auf einen nicht vorhandenen Verzeichnispfad befüllt, wird die Konfigurationsdatei im aktuellen Ausführungsverzeichnis von **GINA2010** gesucht.

Wird keine Konfigurationsdatei gefunden, werden die Angaben zu **Communication Types** in `io_initiate` ohne ein Routing direkt verwendet.

Diese Umgebungsvariable wird bei jedem Aufruf der **Platform Adapter**-Funktion `io_initiate` ausgelesen.

3.5.2 Logging über den Platform Adapter

Die hier aufgeführten Umgebungsvariablen steuern die Generierung, Komprimierung, Namensgebung und Freigabe (Löschen) der vom **Platform Adapter** verwalteten Log-Dateien.

3.5.2.1 GDI_PALOG_DIR (Logdatei-Verzeichnis)

Ist diese Variable verfügbar, so speichert der **Platform Adapter** Log-Dateien, welche mittels `os_openDebug` geöffnet und mittels `os_writeDebug` befüllt werden, in das hier angegebene Zielverzeichnis.

- Ist der angegebene Pfad ein relativer Pfad, so richtet sich dieser am aktuellen Ausführungsverzeichnis von **GINA2010** aus.
- Ist diese Umgebungsvariable leer oder nicht gesetzt, werden keine Log-Dateien generiert.

	Das Zielverzeichnis muss existieren. Es wird nicht automatisch angelegt. Ist das Verzeichnis nicht vorhanden, schlägt `os_openDebug` fehl.
	Der Dateipfad einer Log-Datei darf nicht länger als 256 Zeichen sein. Es ist möglich, dass der Name einer Log-Datei bis zu 100 Zeichen enthalten kann.
	Diese Umgebungsvariable wird bei jedem Aufruf der **Platform Adapter**-Funktion `os_openDebug` ausgelesen.

3.5.2.2 GDI_PALOG_LEN (Logdatei-Länge)

Diese Umgebungsvariable beinhaltet die Ziel-Länge von Log-Dateien, welche mittels der **Platform Adapter**-Funktionen `os_openDebug` geöffnet und mittels `os_writeDebug` befüllt werden. Es wird eine Ganzzahl erwartet, welche die Ziel-Länge der Log-Dateien festlegt. Ist die Ziel-Länge abzüglich der Kopfinformationen überschritten, werden Umbruchdateien gebildet. Bei deaktivierter Komprimierung [3.5.2.4] wird eine ihre Ziel-Länge überschreitende Log-Datei `[LogName].log` in `[LogName].bak.log` umbenannt. Eine neue Log-Datei `[LogName].log` wird angelegt und im weiteren Verlauf bis zum nächsten Umbruch beschrieben.

Bereits vorhandene Umbruchdateien werden bei einem weiteren Umbruch überschrieben. Mit der Einstellung einer Ziellänge über diese Umgebungsvariable können somit Log-Inhalte verlorengehen. Ist dies nicht gewünscht, sollte eine Einstellung der Ziel-Länge in Kombination mit der Komprimierung [3.5.2.4] von Log-Dateien verwendet werden, bei welcher statt des Überschreibens die Komprimierung erfolgt.

Bei aktivierter Komprimierung und aktivierter Ziellänge entstehen andere Dateinamen vor dem Hintergrund einer kollisions- und somit verlustfreien Ablage der Log-Inhalte. Die Bildung der Dateinamen, umgebrochener und komprimierter Dateien ist im Kapitel 3.5.2.4 beschrieben.

- Der Inhalt der Variablen wird als nicht vorzeichenbehaftet Ganzzahl interpretiert. Die aktuelle Interpretation wird dem Systemaufruf `atoi` überlassen und im Kopf einer jeden Log-Datei eingetragen.

- Ist diese Umgebungsvariable gleich `0`, leer oder nicht gesetzt, gibt es keine Längenbegrenzung für Log-Dateien, sodass keine Umbruch-Dateien gebildet werden.

Diese Umgebungsvariable wird bei jedem Aufruf der **Platform Adapter**-Funktion `os_openDebug` ausgelesen.

3.5.2.3 GDI_PALOG_NAMESTAMP (Logdatei-Name)

Diese Umgebungsvariable legt das Format des Dateinamens der Log-Dateien fest, welche der **Platform Adapter** bei Aufruf seiner Funktion `os_openDebug` generiert.

- Ist der Inhalt dieser Umgebungsvariable `"TIME"`, werden das aktuelle Datum und die aktuelle Uhrzeit sekunden-genau in den Dateinamen in der Form `[LogName]_YYYYMMDD_hhmmss.log` aufgenommen.

  Damit werden bei Angabe desselben Namens bei mehrmaligen Aufrufen von `os_openDebug` (nicht innerhalb derselben Sekunde) verschiedene Dateien erzeugt.

- Ist der Inhalt dieser Umgebungsvariablen `"DATE"`, wird nur das aktuelle Datum tag-genau in den Dateinamen in der Form `[LogName]_YYYYMMDD.log` aufgenommen.

  Damit wird bei Angabe desselben Namens bei mehrmaligen Aufrufen von `os_openDebug` an einem Tag nur eine Log-Datei erzeugt und ihr Inhalt ständig erweitert.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, dann wird nur der beim Aufruf von `os_openDebug` übergebene Name verwendet. Damit wird bei Angabe desselben Namens bei mehrmaligen Aufrufen von `os_openDebug` nur eine Log-Datei erzeugt und ihr Inhalt ständig erweitert.

Diese Umgebungsvariable wird bei jedem Aufruf der Platform Adapter-Funktion os_openDebug ausgelesen.

3.5.2.4 GDI_PALOG_ZIP (Logdatei-Komprimierung)

Diese Umgebungsvariable aktiviert bzw. deaktiviert die Komprimierung geschlossener Log-Dateien (auch Umbruch-Dateien), welche der **Platform Adapter** bei Aufrufen seiner Funktion `os_openDebug` angelegt hatte, nun aber nicht mehr beschreibt.

Die Komprimierung nimmt der **Platform Adapter** im Hintergrund mit verminderter Priorität vor, um die Abläufe im Softwaresystem möglichst wenig zu behindern.

- Ist der Inhalt dieser Umgebungsvariable `"true"`, werden nicht mehr zu beschreibende Log-Dateien in 7zip-Archivdateien (`*.7zip`, `*.zip`) komprimiert.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, dann wird die Komprimierung von Log-Dateien deaktiviert.

Bei aktivierter Komprimierung werden die Dateinamen von Umbruchdateien, welche bei Aktivierung der Ziel-Länge von Log-Dateien via GDI_PALOG_LEN [3.5.2.2] gebildet werden, um den Umbruch-Postfix `„_U_YYYYMMDD_hhmmss.log"` anstelle von `„.bak.log"` erweitert.

Bei aktivierter Komprimierung werden diese Umbruchdateien und geschlossene Log-Dateien, die auch zukünftig aufgrund ihrer Namensgebung [3.5.2.3] nicht mehr zum Anhängen von Log-Inhalten geöffnet werden, komprimiert. Die Dateinamen dieser Dateien, werden um den Komprimierungs-Postfix „_C_YYYYMMDD_hhmmss.log.[compext]“ erweitert und mittels 7zip-Mechanismus komprimiert. Die Dateierweiterung [compext] komprimierter Dateien ist dabei vom Betriebssystem abhängig.

Diese Umgebungsvariable wird regelmäßig im Hintergrund ausgelesen.

3.5.2.5 GDI_PA_MEM_LEVEL (Logdatei-Freigabe)

Diese Umgebungsvariable aktiviert bzw. deaktiviert das Löschen komprimierter Log-Dateien und Umbruchdateien in Abhängigkeit des noch verfügbaren Speichers der Festplatten-Partition, auf welcher das Log-Verzeichnis liegt.

Wenn der verfügbare Speicher kleiner als der hier eingestellte Wert in Megabyte ist, wird die Lösch-Anforderung aktiv und komprimierte Dateien werden gelöscht, so lange, bis der verfügbare Speicher größer-gleich GDI_PA_MEM_LEVEL + GDI_PA_MEM_FREE ist und somit mindestens der Freigabe-Bereich wieder verfügbar ist.

Enthält die Umgebungsvariable GDI_PA_MEM_FREE keine gültige Ganzzahl, so wird statt dessen mit 25% des Wertes aus GDI_PA_MEM_LEVEL gerechnet.

<table>
<tr><td colspan="3">gesamter Speicher</td></tr>
<tr><td colspan="2">belegter Speicher (vorher)</td><td>verfügbarer Speicher</td></tr>
<tr><td></td><td>GDI_PA_MEM_FREE</td><td>GDI_PA_MEM_LEVEL</td></tr>
<tr><td>zugesicherter Bereich</td><td colspan="2">Freigabe-Bereich</td></tr>
<tr><td></td><td></td><td>Lösch-Anforderung</td></tr>
<tr><td>Belegter Speicher (nachher)</td><td colspan="2">verfügbarer Speicher</td></tr>
</table>

Tabelle 1: Löschen komprimierter Log-Dateien (Beispiel Löschanforderung)

- Ist der Inhalt der Umgebungsvariablen eine gültige Ganzzahl, so wird das Löschen komprimierter Dateien aktiviert.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, werden keine komprimierten Dateien gelöscht. In diesem Fall ist der Inhalt der Umgebungsvariablen `GDI_PA_MEM_FREE` nicht von Bedeutung.

Dieser Mechanismus arbeitet zusätzlich neben dem Mechanismus zum altersabhängigen Löschen [3.5.2.7] von Log-Dateien.

Diese Umgebungsvariable wird regelmäßig im Hintergrund ausgelesen.

3.5.2.6 GDI_PA_MEM_FREE (Logdatei-Freigabe)

Diese Umgebungsvariable steuert in Verbindung mit der Umgebungsvariablen `GDI_PA_MEM_LEVEL` das Löschen komprimierter Log-Dateien. Der Mechanismus ist entsprechend im Kapitel 3.5.2.5 beschrieben.

- Ist der Inhalt der Umgebungsvariablen eine gültige Ganzzahl, so steuert diese zusammen mit dem Wert aus `GDI_PA_MEM_LEVEL` den Freigabebereich [3.5.2.5] beim Löschen komprimierter Log-Dateien.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, dann steuert der Wert aus `GDI_PA_MEM_LEVEL` den Freigabebereich allein.

Diese Umgebungsvariable wird regelmäßig im Hintergrund ausgelesen.

3.5.2.7 GDI_PALOG_AGE (Logdatei-Freigabe)

Diese Umgebungsvariable steuert das Löschen von komprimierten Log-Dateien in Abhängigkeit von deren Alter.

- Ist der Inhalt der Umgebungsvariablen eine gültige Ganzzahl, so wird diese als das maximale Alter der komprimierten Log-Datei interpretiert. Ist die Datei älter als dieser Wert in Minuten, wird sie von der Festplatte entfernt.

- Wenn die Variable nicht gesetzt oder der Inhalt unbekannt ist, dann werden komprimierte Log-Dateien nicht in Abhängigkeit von ihrem Alter gelöscht.

	Dieser Mechanismus arbeitet zusätzlich neben dem Mechanismus zum Löschen von Log-Dateien in Abhängigkeit vom verfügbaren Speicher der Partition [3.5.2.5].
	Diese Umgebungsvariable wird regelmäßig im Hintergrund ausgelesen.

3.5.3 Logging des Coordinator Engine Protokolls

Die hier aufgeführten Umgebungsvariablen steuern das **Coordinator Engine** Protokoll Logging der Aufrufe zwischen **Coordinator Engine** und **Device Drivers** beidseitig.

Das Logging der **Coordinator Engine** wird über den **Platform Adapter** realisiert und unterliegt somit zusätzlich den Einstellungen aus Kapitel 3.5.1 zu Ablageverzeichnis, Namensgebung, Komprimierung und Freigabe von Log-Dateien.

3.5.3.1 SetGDIEngineProtocol (Log-Aktivierung)

Diese Umgebungsvariable aktiviert bzw. deaktiviert das Logging der Aufruf von API-Funktionen von **Device Drivers** durch die **Coordinator Engine**.

- Ist der Inhalt dieser Umgebungsvariable `"ON"`, wird das Logging unter zusätzlicher Berücksichtigung untenstehender Hinweise aktiviert.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit anderem Inhalt gefüllt, dann wird das Logging deaktiviert.

	Das Logging wird nur dann aktiviert, wenn zusätzlich ein gültiges Logdatei-Verzeichnis via [3.5.2.1] gesetzt wurde.
	Das Logging wird nur dann aktiviert, wenn zusätzlich ein Name für die Logdaten-Senke [3.5.3.2] angegeben wurde.
	Durch zusätzliche Aktivierung des Loggings für Parameter [3.5.3.3] kann die Log-Tiefe erweitert werden.
	Diese Umgebungsvariable wird beim ersten Laden eines Gerätetreibers ausgelesen.

3.5.3.2 GDIEngineProtocolFileName (Logdatei-Name)

Diese Umgebungsvariable steuert den Namen der Logdatei für das **Coordinator Engine** Protokoll Logging. Die Datei wird im Logverzeichnis des **Platform Adapters** und entsprechend allen weiteren Einstellungen laut Kapitel 3.5.1 unter Verwendung des angegebenen Namens gespeichert.

	Das Logging wird nur dann aktiviert, wenn zusätzlich ein gültiges Logdatei-Verzeichnis [3.5.2.1] gesetzt wurde.
	Das Logging wird nur dann aktiviert, wenn zusätzlich eine Aktivierung entsprechend Kapitel 3.5.3.1 vorgenommen wurde.
	Durch zusätzliche Aktivierung des Loggings für Parameter [3.5.3.3] kann die Log-Tiefe erweitert werden.
	Diese Umgebungsvariable wird beim ersten Laden eines Gerätetreibers ausgelesen.

3.5.3.3 SetGDIEngineDataMonitor (Log-Tiefe)

Diese Umgebungsvariable stellt die Log-Tiefe des **Coordinator Engine** Protokoll Loggings ein.

- Ist der Inhalt dieser Umgebungsvariable "ON", werden zusätzlich zu den Funktionsaufrufen in/von **Device Drivers** die übergebenen Ein- und zurückgegebenen Ausgangs-Parameter geloggt.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit anderem Inhalt gefüllt, dann werden Parameter-Inhalte nicht geloggt.

	Das Logging wird nur dann aktiviert, wenn zusätzlich ein gültiges Logdatei-Verzeichnis [3.5.2.1] gesetzt wurde.
	Das Logging wird nur dann aktiviert, wenn zusätzlich eine Aktivierung entsprechend Kapitel 3.5.3.1 vorgenommen wurde.
	Das Logging wird nur dann aktiviert, wenn zusätzlich ein Name für die Logdaten-Senke [3.5.3.2] angegeben wurde.
	Diese Umgebungsvariable wird beim ersten Laden eines Gerätetreibers ausgelesen.

3.5.4 Logging der Coordinator Engine API

Die hier aufgeführten Umgebungsvariablen steuern das **Coordinator Engine** API Logging der Aufrufe zwischen **Coordinator Engine** und darüber liegenden Softwareschicht (hier **GINA2010**) beidseitig.

Das Logging der **Coordinator Engine** wird über den **Platform Adapter** realisiert und unterliegt somit zusätzlich den Einstellungen aus Kapitel 3.5.1 zu Ablageverzeichnis, Namensgebung, Komprimierung und Freigabe von Log-Dateien.

3.5.4.1 GDIEngineAPIProtocolFileName (Logdatei-Name)

Diese Umgebungsvariable steuert den Namen der Logdatei das **Coordinator Engine** API Logging. Die Datei wird im Logverzeichnis des **Platform Adapters** und entsprechend allen weiteren Einstellungen laut Kapitel 3.5.1 unter Verwendung des angegebenen Namens gespeichert.

Das Logging wird nur dann aktiviert, wenn zusätzlich ein gültiges Logdatei-Verzeichnis [3.5.2.1] gesetzt wurde.

Diese Umgebungsvariable wird beim ersten Laden eines Gerätetreibers ausgelesen.

3.5.5 Ausnahmebehandlung

3.5.5.1 GDI_PA_HANDLE_EXCEPTION (Platform Adapter)

Diese Umgebungsvariable aktiviert bzw. deaktiviert die Ausnahmebehandlung via `try` und `catch` durch den **Plattform Adapters** über API-Aufrufe in **Platform Adapter Extensions** und Callback-Aufrufe in **Device Drivers**. Ist die Ausnahmebehandlung aktiviert, werden Ausnahmen nicht in höhere Softwareschichten signalisiert.

- Wird diese Umgebungsvariable auf „`false`" gesetzt, behandelt der **Plattform Adapter** auftretende Ausnahmen aus oben genannten Aufrufen nicht.
- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, werden Ausnahmen aus oben genannten Aufrufen behandelt.

Diese Umgebungsvariable wird nur beim ersten Aufruf einer beliebigen **Plattform Adapter**-Funktion ausgelesen.

3.5.5.2 SetGDIEngineExceptionHandling (Coordinator Engine)

Diese Umgebungsvariable aktiviert bzw. deaktiviert die Ausnahmebehandlung der **Coordinator Engine** aus Callback-Aufrufen (**Information Report**, **Accept**) in die Anwendung **GINA2010** und aus API-Aufrufen in **Device Drivers**.

Die Ausnahmebehandlung ist Microsoft-spezifisch mit `__try` und `__except` umgesetzt, und kommt somit unter UNIX-Systemen nicht zur Anwendung.

- Wird diese Umgebungsvariable auf „`OFF`" gesetzt, behandelt die **Coordinator Engine** auftretende Ausnahmen aus oben genannten Aufrufen nicht.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit anderem Inhalt gefüllt, werden Ausnahmen aus oben genannten Aufrufen behandelt, ausgenommen unter UNIX-Systemen.

Die Einstellung wird während des Ladens eines jeden **Device Drivers** interpretiert und ist für jeden **Device Driver** getrennt einstellbar.

3.5.6 Fehlersuche

3.5.6.1 GDI_PA_MSG_ON_EXCEPTION (Platform Adapter)

Diese Umgebungsvariable aktiviert bzw. deaktiviert Benutzermeldungen für von **Platform Adapter Extensions** geworfene Ausnahmen. Voraussetzung für eine Aktivierung der Benutzermeldungen ist die Aktivierung der Ausnahmebehandlung [3.5.5].

Sind Benutzermeldungen aktiviert und es tritt eine Ausnahme auf, so wird unter Windows-Systemen eine blockierende Messagebox angezeigt. Unter Unix-Systemen erfolgt eine nicht blockierende Fehler-Ausgabe auf der Konsole.

- Ist der Inhalt dieser Umgebungsvariablen `„true“` und tritt eine Ausnahme bei einem Aufruf einer **Platform Adapter Extension**-Funktion auf, wird eine Benutzermeldung angezeigt.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, werden Meldungen über Ausnahmen nicht angezeigt.

Diese Umgebungsvariable wird nur bei Auftreten der ersten Ausnahme ausgelesen.

Unter Windows-System erscheint bei Auftreten einer Ausnahme eine Messagebox. Diese blockiert den aktuellen Steuerfluss bis zur Bestätigung der Messagebox.

Benutzermeldungen aus nebenläufigen Steuerflüssen können wichtige Abläufe stören oder gar Abstürze provozieren. Sie sollten ausschließlich für spezielle Durchläufe zur Fehlersuche aktiviert werden.

3.5.6.2 GDI_PAEXT_IP_MSG_ON_ERROR (IP-Verbindung)

Diese Umgebungsvariable aktiviert bzw. deaktiviert Benutzermeldungen für Fehler bei der Verwendung von IP-Sockets zur Kommunikation über Ethernet-Kanäle.

Sind Benutzermeldungen aktiviert und es tritt ein IP-Kommunikationsfehler auf, so wird unter Windows-Systemen eine blockierende Messagebox angezeigt. Unter Unix-Systemen erfolgt eine nicht blockierende Fehler-Ausgabe auf der Konsole.

- Ist der Inhalt dieser Umgebungsvariablen „true“ und tritt ein IP-Kommunikationsfehler bei der Kommunikation via IP auf, wird eine Benutzermeldung angezeigt.
- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, werden IP-Kommunikationsfehler nicht angezeigt.

	Diese Umgebungsvariable wird bei jedem Aufruf der **Platform Adapter**-Funktion `io_initiate` zur Initialisierung von Ethernet-Verbindungen ausgelesen.
	Unter Windows-System erscheint bei Auftreten eines IP- Kommunikationsfehlers eine Messagebox. Diese blockiert den aktuellen Steuerfluss bis zur Bestätigung der Messagebox.
	Benutzermeldungen aus nebenläufigen Steuerflüssen können wichtige Abläufe stören oder gar Abstürze provozieren. Sie sollten ausschließlich für spezielle Durchläufe zur Fehlersuche aktiviert werden.

3.5.6.3 GDI_PAEXT_COM_MSG_ON_ERROR (Serielle Verbindung)

Diese Umgebungsvariable aktiviert bzw. deaktiviert Benutzermeldungen für Fehler bei der Kommunikation über serielle Kanäle (COM).

Sind Benutzermeldungen aktiviert und es tritt ein COM-Kommunikationsfehler auf, so wird unter Windows-Systemen eine blockierende Messagebox angezeigt. Unter Unix-Systemen erfolgt eine nicht blockierende Fehler-Ausgabe auf der Konsole.

- Ist der Inhalt dieser Umgebungsvariablen „true“ und tritt ein Fehler bei einer seriellen Kommunikation auf, wird eine Benutzermeldung angezeigt.

- Ist diese Umgebungsvariable leer, nicht gesetzt oder mit nicht spezifiziertem Inhalt gefüllt, werden Kommunikationsfehler nicht angezeigt.

	Diese Umgebungsvariable wird bei jedem Aufruf der **Platform Adapter**-Funktion `io_initiate` zur Initialisierung serieller Verbindungen (COM) ausgelesen.
	Unter Windows-System erscheint bei Auftreten eines COM-Kommunikationsfehlers eine Messagebox. Diese blockiert den aktuellen Steuerfluss bis zur Bestätigung der Messagebox.
	Benutzermeldungen aus nebenläufigen Steuerflüssen können wichtige Abläufe stören oder gar Abstürze provozieren. Sie sollten ausschließlich für spezielle Durchläufe zur Fehlersuche aktiviert werden.

3.6 Platform Adapter Log-Dateien

Wird ein **RDE Platform Adapter** verwendet, so können Log-Einträge ein einem bestimmten Format erzeugt werden.

Entsprechend den gesetzten Umgebungsvariablen werden die Log-Einträge in Log Dateien geschrieben.

Die Log-Einträge haben folgendes Format: `"[HND@GDI4.X][THREAD][hh:mm:ss.SSS]"`.

Die einzelnen Elemente haben die hier aufgelistete Bedeutung:

- HND

  GDI-Handle, welches bei `os_openDebug` vergeben wurde.

- X

  GDI-Version der verwendeten `os_writeDebug` Funktion.

- THREAD

  Ist die Thread-Identifikationsnummer des aufrufenden Threads. Unter Windows wird zur Ermittlung der Thread-ID die Systemfunktion `GetCurrentThreadId` verwendet, unter Unix-System der Aufruf `pthread_self`.

Danach folgt der Inhalt, welcher vom Aufrufer an `os_writeDebug` übergeben wurde. Jeder Eintrag wird mit einem Zeilenumbruch abgeschlossen.

4 Device Driver

Device Drivers liegen üblicherweise als ANSI C/C++ Quellcode vor und werden auf der Zielplattform übersetzt.

Es ist aber auch möglich, sie auf kompatiblen Systemen zu erstellen und zu liefern.

4.1 Alignments

Alignments werden in den folgenden Kapiteln für drei verschiedene Kategorien festgelegt.

4.1.1 Alignment der typisierten API-Strukturen

Typisierte API-Strukturen sind alle definierten Strukturen, welche vollständig typisiert in API-Funktionen von Standardkomponenten nach **ASAM GDI** verwendet werden und nicht letztendlich durch das generische Typkonstrukt `void*` beschrieben werden.

Diese Strukturen sind bei Erstellung der zugehörigen Softwarekomponenten mit dem richtigen Alignment zu übersetzen.

- Auf Windows-Systemen sind alle API-Strukturen der mit **GINA2010** gelieferten Bibliotheken mit einem Alignment von `8` erstellt.

- Auf Unix-Systemen sind alle API-Strukturen der mit **GINA2010** gelieferten Bibliotheken ohne Angabe eines Alignments erstellt.

  Auf Unix-Systemen und bei Verwendung einer IA32-Architektur ist üblicherweise ein Alignment von `4` vorzufinden.

Auf Unix-Systemen wird generell die Erstellung aller Komponenten direkt auf dem Zielsystem empfohlen, um Alignment-Konflikten vorzubeugen.

⚠	Hersteller weiterer nicht mit **GINA2010** gelieferten **Platform Adapter Extensions** müssen zwingend die Alignment-Festlegungen für feste API-Strukturen einhalten und die ihnen zugehörige **ASAM GDI** Schnittstelle für **Platform Adapter Extensions** entsprechend übersetzen.
⚠	Hersteller von **Device Drivers** müssen zwingend die Alignment-Festlegungen für feste API-Strukturen einhalten und die ihnen zugehörige **ASAM GDI** Schnittstelle für **Device Drivers** entsprechend übersetzen.

4.1.2 Alignment der Communication Type Strukturen

Communication Types sind spezielle Kanäle aus **Platform Adapter Extensions**, um welche die vorliegende **GINA2010**-Installation erweitert werden kann, um für **Device Drivers** weitere Arten von Kommunikations-Kanälen oder anderweitige zusätzliche Systemressourcen verfügbar zu machen. Die durch einen **Communication Type** definierten Strukturen werden vorerst allgemein durch das generische Typkonstrukt `void*` beschrieben und an der Schnittstelle von **Platform Adapters** und **Platform Adapter Extensions** eingesetzt. Die Festlegung der Typen erfolgt durch Definition des **Communication Type**.

Tabelle 2 zeigt am Beispiel Typisierungen bei Verwendung der erweiternden Funktion `"ReadEx"` des **Communication Type** `"GDI_EIP:3.0"` der mit **GINA2010** gelieferten **Platform Adapter Extension** `rde_paext_std_433.dll`. Die zugehörigen Definitionen sind in den Headerdateien `gdi_paextcttype10.h` und `gdi_paextcttype10.h` zu finden.

API-Funktionen **gdi_pa.h** **gdi_paext.h**	**Funktion** **gdi_paextcttype10.h**	**Parameter** **gdi_pa.h** **gdi_paext.h**	**Communication Type Struktur** **gdi_paextcttype10.h**
`io_open` `ext_open`		`ptConfigData` `->pvParam`	`APE_IP4_IOPARAM_EIP`
`io_execute` `ext_execute`	„ReadEx“	`pvInput`	`APE_IP4_INPARAM_EIP_READEX`
`io_execute` `ext_execute`	„ReadEx“	`pvOutput`	`APE_IP4_OUTPARAM_EIP_READEX`
`io_execute` `ext_execute`	„ReadEx“	`pvFuncRet`	`APE_IP4_RETPARAM_EIP_READEX`

Tabelle 2: Beispiel einer Communication Type Struktur

Diese Strukturen sind bei Erstellung zugehöriger **Platform Adapter Extensions** und **Device Drivers** mit dem richtigen Alignment zu übersetzen.

- Auf Windows-Systemen sind alle **Communication Types** der mit **GINA2010** gelieferten **Platform Adapter Extensions** mit einem Alignment von `8` erstellt.

- Auf Unix-Systemen sind alle **Communication Types** der mit **GINA2010** gelieferten **Platform Adapter Extensions** ohne Angabe eines Alignments erstellt.

  Auf Unix-Systemen und bei Verwendung einer IA32-Architektur ist üblicherweise ein Alignment von `4` vorzufinden.

	Auf Unix-Systemen wird generell die Erstellung aller Komponenten direkt auf dem Zielsystem empfohlen, um Alignment-Konflikten vorzubeugen.
	Hersteller weiterer nicht mit **GINA2010** gelieferten **Platform Adapter Extensions** müssen zwingend die Alignment-Festlegungen für **Communication Types** einhalten und die so verwendeten Strukturen entsprechend übersetzen.
	Hersteller von **Device Drivers** müssen zwingend die Alignment-Festlegungen für **Communication Types** einhalten und die so verwendeten Strukturen entsprechend übersetzen.

4.1.3 Alignment der DCD-Strukturen

DCD-Strukturen sind Strukturen, welche sich erst aus den Inhalten der **DCD**-Dateien ergeben. Sie werden an den API-Funktionen der **Device Drivers** durch das generische Typkonstrukt `void*` beschrieben.

API-Funktion `gdi_dd.h`	**Parameter** `gdi_dd.h`
`GDI_Execute`	`pvDatPtrW`
`GDI_Execute`	`pvDatPtrR`

Tabelle 3: Beispiel eines generischen Typs zur Aufnahme einer DCD-Struktur

Diese Strukturen sind bei Erstellung zugehöriger **Device Drivers** mit dem gewählten Alignment zu übersetzen.

Der **Coordinator Engine** werden sie erst zur Laufzeit mit Einlesen der **DCD**-Dateien bekannt.

Der Tatsache geschuldet, dass die in **GINA2010** integrierte **Coordinator Engine** mit verschiedenen **Device Drivers** über verschiedene **DCD**-Strukturen kommuniziert, welche der **Coordinator Engine** erst durch Einladen der jeweils einem **Device Driver** zugehörigen **DCD** bekannt werden, benötigt die **Coordinator Engine** die Information über das vom jeweiligen **Device Driver** verwendete Alignment bei dessen Erstellung. Hierbei bestimmt der **Device Driver** das verwendete Alignment.

Das Alignment von **DCD**-Strukturen ist in **GINA2010** vom Benutzer beim Laden [5.5.2] eines **Device Driver** anzugeben.

Ist der betreffende **Device Driver** nach **GDI**-Version 4.5 oder höher implementiert, liefert er das zu verwendende Alignment seiner **DCD**-Strukturen über die API-Funktion `GDI_Attach`, nach dessen Aufruf **GINA2010** die Benutzervorgabe mit dem so gelieferten Alignment überlagert.

	Das Alignment der **DCD**-Strukturen wird vom **Device Driver** vorgegeben.
	**Device Driver** ab **GDI**-Version 4.5 müssen das Alignment im Rückgabewert von `GDI_Attach` liefern.
	Für **Device Driver** bis einschließlich **GDI**-Version 4.4 muss der Benutzer das richtige Alignment beim Laden des **Device Drivers** einstellen.

4.2 Installation

Die Benennung von **Device Drivers** ist genauso frei wählbar wie das Ablageverzeichnis, da die Ladung eines **Device Driver** mit absoluter oder auch relativer Pfadangabe erfolgen kann.

Unter Unix-Systemen wird dennoch empfohlen, den in Kapitel 3.2.3 enthaltenen Installationshinweisen zu folgen, um eine konsistente Umgebung aufzubauen.

4.3 Besonderheiten bei Windows

Device Drivers, die mit **RDE** Produkten zusammen unter Windows verwendet werden sollen, müssen folgende Aufrufkonventionen beachten.

Die Funktionen des **Device Driver** werden ohne jede Dekoration exportiert.

Die Windows Aufrufkonvention der exportierten Funktionen ist `__cdecl`.

Die Aufrufkonvention der CallBack-Funktionen ist ebenfalls `__cdecl`. Die Dekoration der Funktionsnamen spielt in diesem Falle keine Rolle, weil die Funktionen über übergebene Zeiger aufgerufen werden und nicht über Exporte.

Vergleiche dazu Kapitel 3.1.3.

5 GINA2010 – Programmbeschreibung

Dieser Abschnitt gibt einige Hinweise zum Programm selbst und dessen Aufbau.

5.1 Programmüberblick

Die Anwendung **GINA2010** kann durch argumentloses Ausführen (z.B. Doppelklick) direkt gestartet werden.

Zum Zeitpunkt des Programmstarts erscheint für ein paar Sekunden ein Intro im Vordergrund, welches man durch Anklicken auch sofort ausblenden kann.

Es erscheint das Hauptfenster der Anwendung.

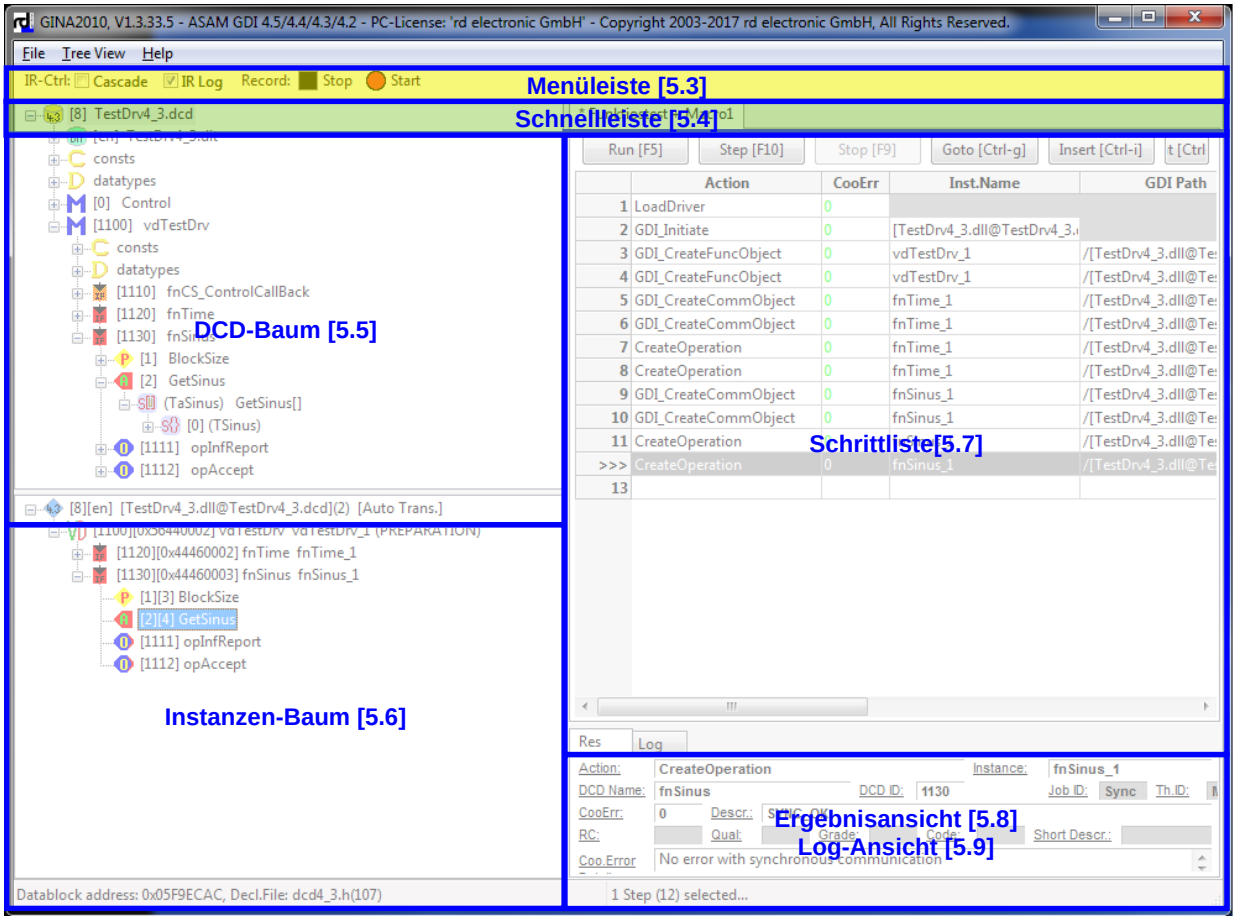


Bild 2: GINA2010 – Hauptfenster

Das Programm besteht aus vier Hauptteilen. Diese befinden sich in den vier Quadranten des Programmfensters. Die Trennlinie (Sash) zwischen den einzelnen Teilen kann beliebig verschoben werden, um die Größe den Erfordernissen anzupassen. Im Folgenden sind diese Teilebereiche beschrieben.

5.2 Allgemeine Funktionen

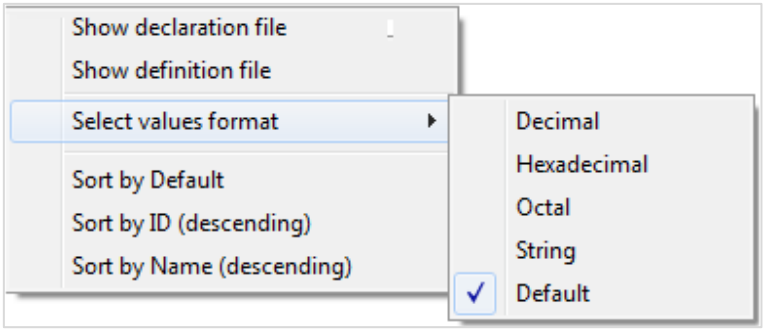


Bild 3: Allgemeine Kontextmenüs

Allgemein unterstützen die Kontext-Menüs der Elemente des **DCD**-Baums, des Instanzen-Baums und die Dialoge zur Eingabe und Anzeige von Datenelementen [5.10] einige brauchbare Funktionen zum Navigieren in den zugeordneten **DCD**-Dateien, zur Sortierung der Elemente im zugehörigen Baum und zum Einstellen des Anzeigeformats von Zahlenwerten.

5.2.1 Navigieren in DCD-Dateien

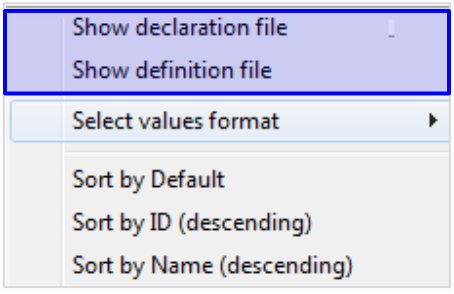


Bild 4: Navigation in DCD-Dateien

Der Menüpunkt Show declaration file öffnet mit dem eingestellten externen Texteditor [5.3.4] die DCD-Datei und setzt den Cursor an die Zeile, in welcher der Typ des ausgewählten Elementes definiert ist.

Der Menüpunkt Show definition file öffnet mit dem eingestellten externen Texteditor [5.3.4] die DCD-Datei und setzt den Cursor an die Zeile, in welcher das ausgewählte Elementes definiert ist.

5.2.2 Anzeige-Format von Zeichen

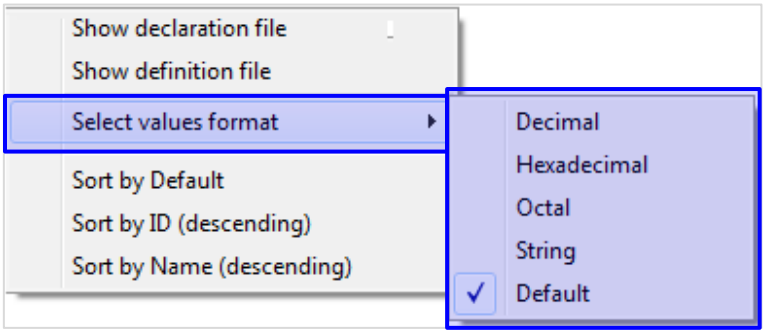


Bild 5: Auswahl des Anzeigformates

Manchmal kann es sinnvoll sein, essentiale Ganzzahltypen in verschiedenen Formaten darzustellen. Unterstützt werden dabei die dezimale, die hexadezimale und die oktale Schreibweise von Ganzzahlen.

Die String-Präsentation ist generell für 8-Bit-Zeichen vorgesehen, kann aber auch ganze Zahlen als mehrstellige Versionszeichenketten mit 1 Byte pro Versionsnummer anzeigen.

Die Default-Einstellung stellt den Datentyp in seinem häufigsten Anzeigeformat dar.

5.2.3 Sortierung der Elemente im Baum

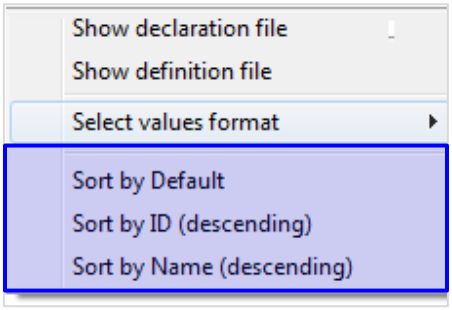


Bild 6: Sortieren von Baumansichten

Viele Elemente der **DCD**-Dateien werden entsprechend der Spezifikation des **ASAM GDI** [GDI-Doc3] mit Identifikationsnummern und Namen versehen. Entsprechend werden hier Aufwärts- und Abwärts-Sortierungen ermöglicht.

5.3 Menüleiste

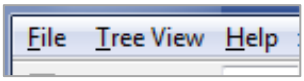


Bild 7: Menüleiste

Dieses Kapitel zeigt die Aktionen auf, welche über die Menüleiste der Anwendung zu erreichen sind.

5.3.1 Öffnen einer DCD

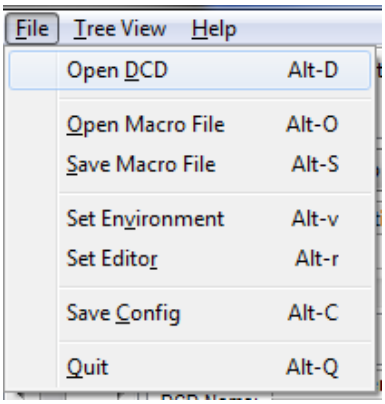


Bild 8: Menü: File→DCD

Über den Menüpunkt File→Open DCD kann eine DCD-Datei geöffnet werden. Hierzu erscheint ein Systemdialog zum Öffnen von Dateien.

Nach Bestätigung dieses Dialogs wird der Inhalt der ausgewählten DCD-Datei in einem neuen Root-Knoten im DCD-Baum [5.5] dargestellt.

Fehler und andere Ausgaben, welche beim Öffnen einer DCD-Datei entstehen, werden in der Log-Ansicht [5.9] sichtbar.

Das Schließen einer DCD-Datei wird im Kontextmenü des DCD-Knotens im DCD-Baum über den Menüpunkt Remove DCD ermöglicht.

Das Modifizieren und Speichern von DCD-Dateien in GINA2010 ist nicht möglich.

5.3.2 Arbeiten mit Makro-Dateien

5.3.2.1 Laden von Makro-Dateien

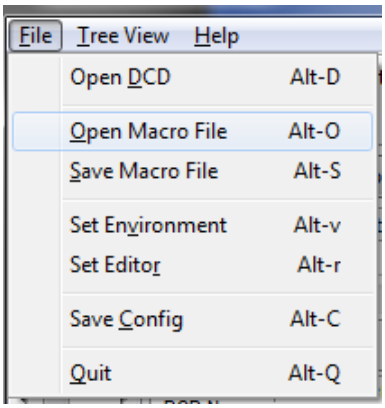


Bild 9: Menü: File→Macro

Über den Menüpunkt File→Open Macro File kann eine Makro-Datei geöffnet werden.

Hierzu erscheint ein Systemdialog zum Öffnen von Dateien.

Nach Bestätigung dieses Dialogs wird der Inhalt der ausgewählten Makro-Datei in die Schrittliste [5.7] ab der dort selektierten Zeile eingefügt.

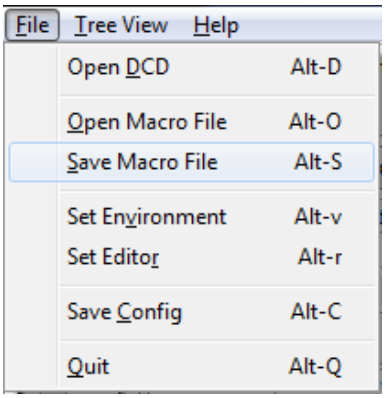
Es ist ebenfalls möglich, die Makro-Datei beim Start von GINA2010 als Parameter anzugeben.

Es ist möglich mehrere, zusammenpassende **Makro**-Dateien hintereinander zu laden. Die **Makro**-Datei wird dabei an die Position der aktuellen Markierung in der Schrittliste [5.7] geladen.

Es zu beachten, dass die **DCD** bereits beim Laden des **Makro**s eingelesen wird. Die **DCD**-Datei muss deshalb bereits jetzt an der angegebenen Position verfügbar sein. Der **Device Driver** wird jedoch erst beim Ausführenden des entsprechenden Schrittes geladen.

5.3.2.2 Speichern des Makros

Das Abspeichern der Schrittliste [5.7] als **Makro**-Datei ist mit dem Menüpunkt Save Macro File des Menüs File möglich. Es wird immer die komplette Schrittliste in der **Makro**-Datei abgespeichert.



Hierzu erscheint ein Systemdialog zum Speichern von Dateien.

Nach Bestätigung dieses Dialogs wird der gesamte Inhalt der Schrittliste [5.7] in die **Makro**-Datei geschrieben.

Es ist zu beachten, dass der Pfad zum **Device Driver** und zur **DCD** in der **Makro**-Datei mit abgespeichert wird. Wenn sich der Pfad ändert ist er in der **Makro**-Datei vor dem Laden anzupassen. Er ist unterhalb der Sektion `[GAT_LOADDRIVER]` in der Datei zu finden.

Die **Makro**-Dateien enden standardmäßig mit der Dateierweiterung „gam". Dies ist jedoch kein Zwang. Es können Dateien mit beliebiger Dateierweiterung geladen werden, solange der Inhalt syntaktisch korrekt ist.

Für geübte Benutzer ist es auch möglich die **Makro**-Dateien mit einem Texteditor zu erstellen.

5.3.2.3 Ausführen des Makros

Das geladene **Makro** wird in der Schrittliste [5.7] dargestellt, kann dort modifiziert und mittels der entsprechenden **Makro**-Funktionen [5.7.2] ausgeführt werden.

5.3.3 Bearbeiten von Umgebungsvariablen

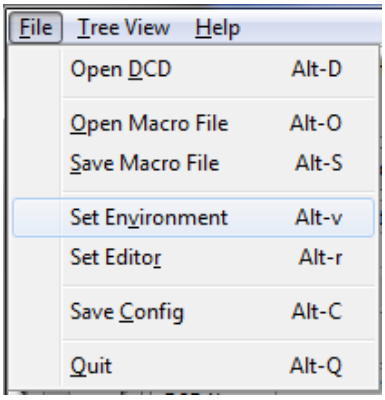


Bild 10: Menü: File→Environment

Über den Menüpunkt File→Set Environment können Umgebungsvariablen für den Prozess hinzugefügt, modifiziert, aktiviert und deaktiviert. Im Kapitel 3.5 befindet sich eine Beschreibung der von **GINA2010** und den darunter liegenden **ASAM GDI** Komponenten der Firma **RDE** verwendeten Umgebungsvariablen.

Bei Auswahl des Menüpunktes Set Environment erscheint ein Dialog dieser Umgebungsvariablen.

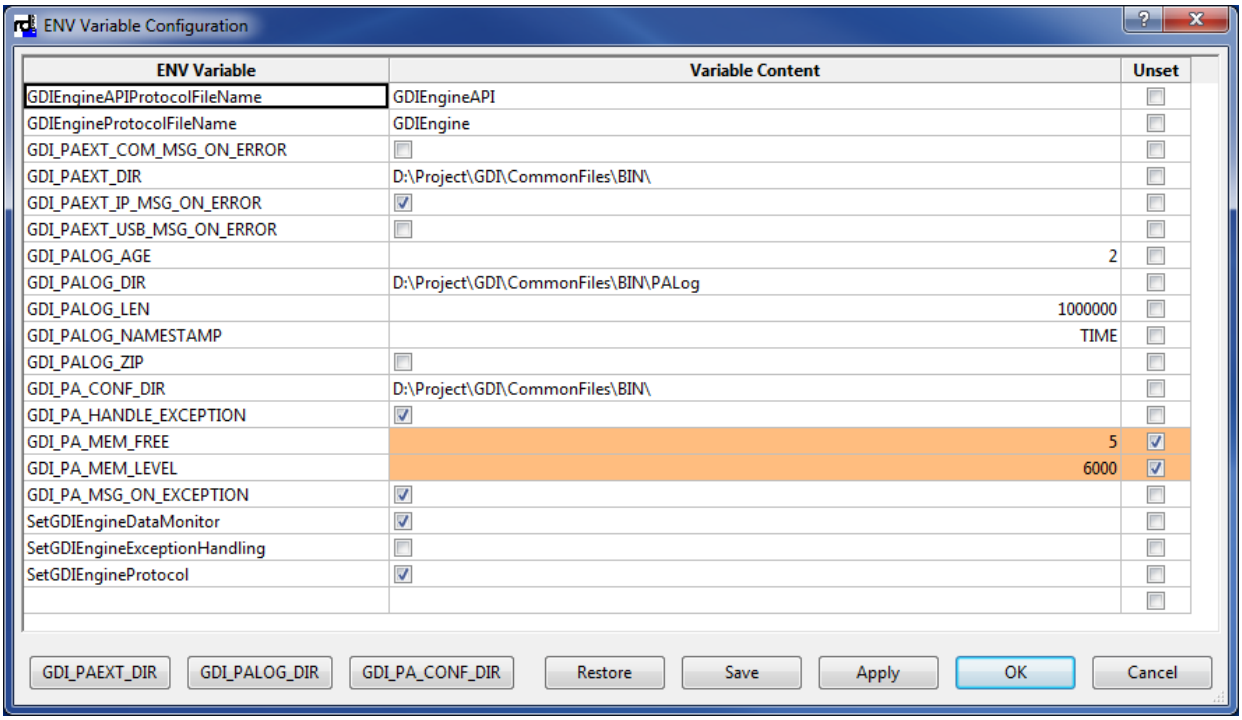


Bild 11: Dialog der Umgebungsvariablen

5.3.4 Einstellen eines externen Text-Editors

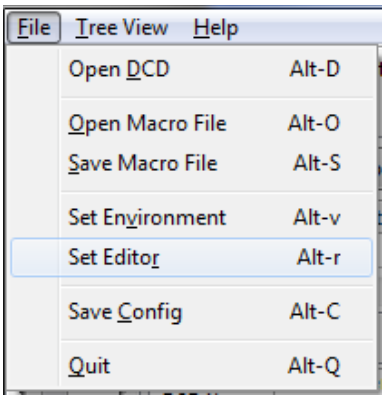


Bild 12: Menü: File → Editor

GINA2010 bietet die Möglichkeit, aus dem **DCD**-Baum [5.5] heraus, die Definitionen und Deklarationen der dort anzeigten Elemente in einem externen Text-Editor gezielt anzuzeigen.

Bei Auswahl des Menüpunktes Set Editor erscheint ein Systemdialog zur Auswahl eines solchen Text-Editors.

Nach Auswahl und Bestätigung erscheint ein Dialog zur Einstellung der zu übergebenden Argumente an den Text-Editor.

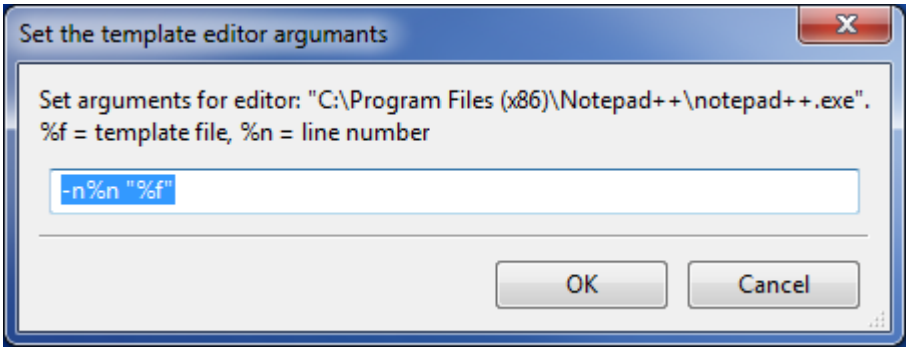


Bild 13: Parametrierung eines externen Text-Editors

Bei Verwendung der Zeichenfolge `"%f"` fügt **GINA2010** den Dateipfad der im späteren Verlauf zu öffnenden Datei und bei Verwendung der Zeichenfolge `„%n"` die Zeilennummer innerhalb dieser Datei an.

Das Beispiel in Bild 13 zeigt eine mögliche Parametrierung des externen Text-Editors notepad++. In welcher Form die Parameter zu übergeben sind, hängt vom verwendeten Text-Editor ab. Der Editor notepad++ definiert beispielsweise als Option `„-n"` zur Vorgabe der Zeilennummer und erwartet als letztes Argument den Dateipfad der zu öffnenden Datei.

Ein einfacher Test der Einstellung ist im Kontextmenü über einem **Interface** im **DCD**-Baum durch Anwahl des Menüpunktes Show declaration file möglich. Der Text-Editor sollte dann die **DCD**-Datei öffnen und den Text-Cursor auf die Zeile der Deklaration des **Interface** setzen.

5.3.5 Speichern der aktuellen GINA-Konfiguration

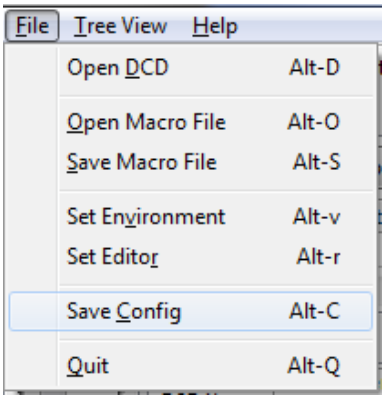


Bild 14: Menü: File->Config

Über das Menü File→Save Config ist es möglich, die aktuellen Einstellungen von **GINA2010** zu sichern.

Gesichert werden:

- Fensterposition
- Maximierung der Anwendung
- Sash Positionen
- **Betriebsmodus** Einstellungen

Diese Einstellungen werden mit den betriessystemtypischen Mechanismen abgelegt (Registry, Config Datei, ...).

Es erfolgt keine automatische Speicherung der Einstellungen beim Beenden von **GINA2010**. Beim nächsten Start ist somit die zuletzt abgespeicherte Konfiguration wieder aktiv.

Die Wiederherstellung der voreingestellten Sash Positionen ist durch Doppelklicken der Trennleisten (Sashs) der Teilbereiche möglich.

5.3.6 Schließen der Anwendung

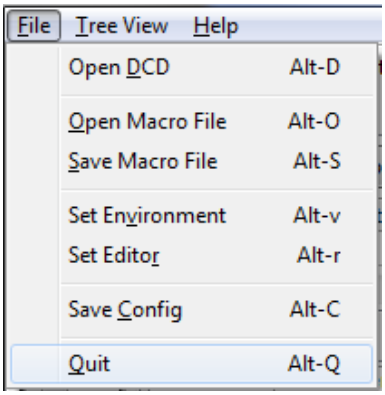


Bild 15: Menü: File->Quit

Durch Auswahl des Menüpunktes Quit wird **GINA2010** beendet.

- Sicherheitsabfrage für ungesicherte Schrittlisten

  Vor dem Beenden prüft GINA2010 ob die Schrittliste in ihrem aktuellen Zustand in eine Datei abgespeichert wurde. Ist dies nicht der Fall, erfolgt eine Sicherheitsabfrage an den Benutzer.

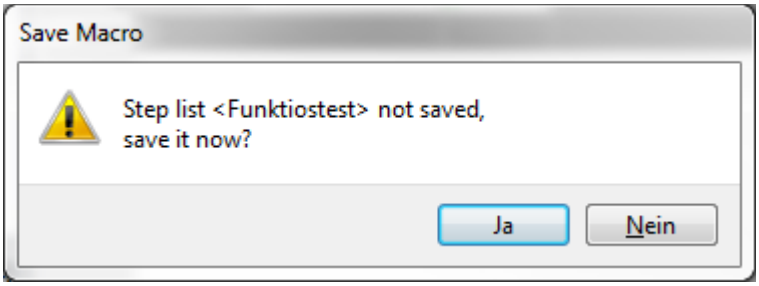


Bild 16: Sicherheitsabfrage: Ungesicherte Schrittliste

Bei Bestätigung dieses Dialogs mittels Schalter Ja öffnet sich ein Systemdialog zum Speichern der Schrittliste [5.3.2]. Der Benutzer kann die Schrittliste nun in einer **Makro**-Datei zur späteren Wiederverwendung speichern.

- Sicherheitsabfrage für ungesicherte Aufzeichnung

  Weiterhin prüft GINA2010 vor dem Beenden ob eine Aufzeichnung aktiv und noch nicht gespeichert wurde. Ist dies nicht der Fall, öffnet sich ein Systemdialog zum Speichern der Aufzeichnungsdatei analog Bild 21 [5.4.2] und der Benutzer kann die Aufzeichnung vor dem Beenden abspeichern.

5.4 Schnellleiste

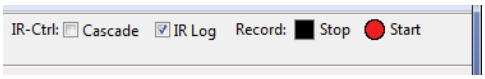


Bild 17: Schnellleiste

Die Schnellleiste enthält Schalter und Eingabefelder zur Ausgabesteuerung von **Information Reports** [5.4.1] und zur Aufzeichnung von Aktionen [5.4.2].

5.4.1 Ausgabesteuerung zu Information Reports

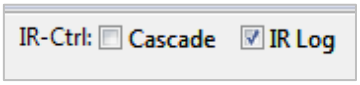


Bild 18: Steuerung der Ausgaben von Information Reports

Mit Cascade kann voreingestellt werden, wie Information Reports grafisch dargestellt werden. Eine Beschreibung ist in Kapitel 5.11 zu finden.

Mit IR Log kann eingestellt werden, dass **GINA2010** die via **Information Reports** von **Device Drivers** übertragenen Daten zur späteren Prüfung in Dateien sichert. Die Parametrierung und das Ausgabe- Format der gesicherten Daten sind im Dokument [RDE-Doc2] beschrieben.

5.4.2 Aufzeichnen von Aktionen (XML-Report)

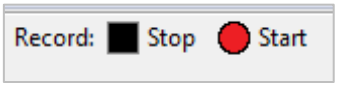


Bild 19: Aufzeichnung inaktiv

Diese Steuerung erlaubt das Aufzeichnen sowohl aller Aktionen, die in der Schrittliste [5.7] erscheinen und ausgeführt werden als auch von Daten, welche via **Information Reports** von **Device Drivers** an **GINA2010** übertragen werden. Diese werden im Folgenden als erfassbare Aktionen bezeichnet.

Über die beiden Schalter Start und Stop wird die Aufzeichnung gestartet bzw. gestoppt.

Beim Starten von **GINA2010** ist die Aufzeichnung zunächst inaktiv [Bild 19]. Es können **Makro**-Dateien geladen und ausgeführt werden. Dabei werden keine Schritte aufgezeichnet und können auch nicht nachträglich aufgezeichnet oder gespeichert werden.

Bei Anwahl des Schalters Start wird die Aufzeichnung aktiviert. Dabei werden zunächst alle Dateinamen (mit Pfad) der verwendeten **DCD**-Dateien (`*.dcd` bzw. `*.h`) und verwendeten **Device-Driver** (`*.dll` bzw. `*.so*`) zusammen mit den dazugehörigen MD5-Checksummen erfasst und im Aufzeichnungs-Speicher von **GINA2010** (Arbeitsspeicher) abgelegt. Eine laufende Aufzeichnung ist am eingerasteten Start-Schalter zu erkennen.

Bild 20: Aufzeichnung aktiv

Alle erfassbaren Aktionen werden ab diesem Zeitpunkt in den Aufzeichnungs-Speicher geschrieben.

Aktionen, welche vor der Aktivierung einer Aufzeichnung ausgeführt wurden, werden nicht aufgezeichnet.

Durch Anwahl des Schalters Stop wird die Aufzeichnung deaktiviert. Der Benutzer wird aufgefordert, einen Dateipfad für die Speicherung der Aufzeichnungs-Datei anzugeben. Das Ausgabe-Format der Aufzeichnungen ist im Dokument [RDE-Doc3] beschrieben.

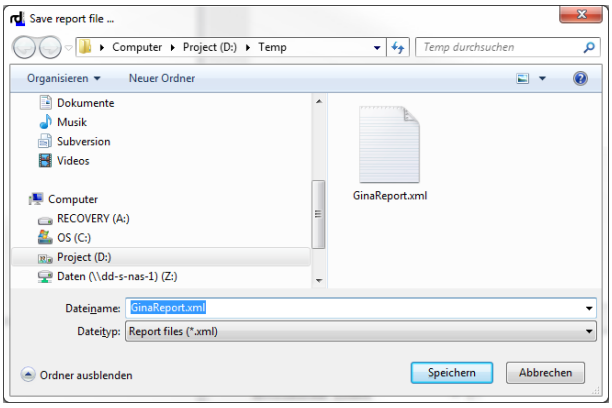


Bild 21: Aufzeichnung: Speichern einer Aufzeichnungs-Datei

Sollte die Aufzeichnung beim Beenden von **GINA2010** noch aktiv sein, erscheint dieser Dialog ebenfalls.

5.5 DCD-Baum

Links oben befindet sich der **DCD**-Baum. Nach Programmstart erscheint hier lediglich ein Icon mit der Beschriftung „No DCD Loaded“ [5.1].

Durch Laden von **DCDs** [5.3.1] oder Laden von **Makro**-Dateien [5.3.2.1] werden die so verwendeten **DCDs** im Baum dargestellt.

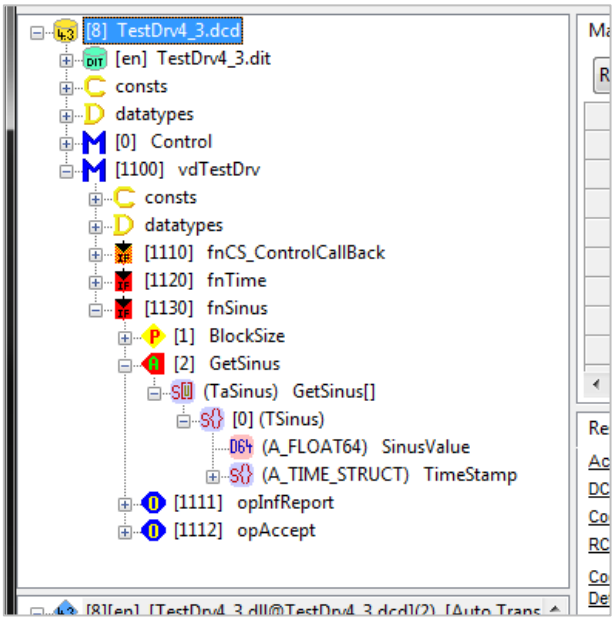


Bild 22: DCD-Baum

Es ist möglich, sowohl die gleiche, als auch verschiedene **DCDs** gleichzeitig zu laden und anzuzeigen. Dabei ist zu beachten, dass **DCDs** im Hintergrund so lange gespeichert bleiben, bis keines der Programmteile mehr darauf referenziert. Das kann dazu führen, dass ein Abfragedialog ob die **DCD** wirklich erneut geladen werden soll auch dann erscheint, wenn anscheinend keine **DCD** mehr geladen ist (**DCD**-Baum ist leer). Wird das erneute Laden dann verneint, so wird die bereits geladene **DCD** wieder angezeigt.

Auf dem **DCD**-Icon ist mit der rechten Maustaste ein Kontextmenüpunkt Remove erreichbar. Damit wird die **DCD** aus dem Baum entfernt, wenn sie nicht mehr referenziert wird.

Mit der Linken Maustaste können Teile der **DCD** durch Ziehen kopiert werden. Dies ist sowohl in diesem Baum als auch im Instanzen-Baum [5.6] möglich. Mit dieser Funktionalität soll das Vergleichen bestimmter Teile bei großen **DCDs** erleichtert werden. Es hat keinen Einfluss auf die Benutzung des **Device Driver** oder dessen Instanzen.

Im **DCD**-Baum stehen ständig die Allgemeinen Funktionen entsprechend Kapitel 5.2 zur Verfügung.

5.5.1 Liste der Symbole

Im Folgenden werden die verwendeten Symbole, die zur Darstellung der **DCD** verwendet werden, kurz beschrieben.

	Eine **DCD** wurde fehlerfrei geparst. Wurde die **GDI** Version erkannt wird ein Icon mit entsprechendem Eintrag verwendet.
	Eine **DCD** wurde mit Warnungen geparst. Das können z.B. nicht verwendete Forward Deklarationen sein. Die genaue Ursache ist der Log-Ansicht [5.9] zu entnehmen. Die Warnung wird dort blau dargestellt. Wurde die **GDI** Version erkannt wird ein Icon mit entsprechendem Eintrag verwendet.
	Eine **DCD** wurde mit Fehlern geparst. Das hat normalerweise eine unvollständige Anzeige zur Folge. Die genaue Ursache ist der Log-Ansicht [5.9] zu entnehmen. Der Fehler wird dort rot dargestellt. Wurde die **GDI** Version trotzdem erkannt wird ein Icon mit entsprechendem Eintrag verwendet.
	Ein Teil einer fehlerfreien **DCD** wurde mit Hilfe von Drag und Drop kopiert.
	Ein Teil einer mit Warnungen geparsten **DCD** wurde mit Hilfe von Drag und Drop kopiert.
	Ein Teil einer fehlerhaften **DCD** wurde mit Hilfe von Drag und Drop kopiert.
	In diesem Zweig sind alle impliziten und expliziten Konstanten aufgelistet. Die-

	sen Eintrag kann es unterhalb der **DCD** selbst und unterhalb eines jeden **Modules** geben.
	**Module** Globale **Interfaces** von **GDI** 4.2 werden in einem virtuellen **Module** mit der ID `1` und dem Namen „Global“ dargestellt. **Transition Function** und **Device Base Function** im **Control VD** mit der ID `0`.
	**Interface** (auch **Device Function** genannt)
	**Abstract Interface** Derartige **Interfaces** können nicht instanziiert werden. Sie dienen anderen, abgeleiteten **Interfaces** als Basisobjekt.
	Parameter Die Symbole kennzeichnen readonly, read/write und writeonly (**Create Parameter**).
	**Attribute** Die Symbole kennzeichnen readonly und read/write **Attribute**
	**Operation** Die Existenz von Eingabe- und/oder Ausgabewerten ist an den unten beschriebenen Symbolen der nächsten Ebene erkennbar.
	Virtuelle **Operation** Diese **Operation** muss in einem abgeleiteten **Interface** implementiert werden und ist selbst nicht benutzbar.
	Eingabe- und Ausgabewerte einer **Operation** Diese Symbole werden nur dargestellt wenn die **Operation** entsprechende Werte hat.
	Integer-Datenelemente mit 8 Bit Breite vom Typ **Octet** und **Char**.
	Integer-Datenelemente mit 16 Bit Breite vom Typ **UShort** und **Short**.
	Integer-Datenelemente mit 32 Bit Breite vom Typ **Ulong** und **Long**.
	Real Datenelemente vom Typ **Float** und **Double** mit 32 oder 64 Bit Breite.
	Datenelement vom Typ **Boolean**. Im **GDI** ist das ein `unsigned` 8-Bit-Wert.
	Datenelement vom Typ **Enumeration**. Im **GDI** ist dass ein 16-Bit-**Short**-Wert. Das rechte Symbol kennzeichnet die verfügbaren **Enumerator** deren Darstellung mit einem Kontextmenü ein und ausgeschaltet werden kann.
	Datenelement vom Typ **Struktur**.
	Datenelement vom Typ **Union**. Der erste Untereintrag ist der **Selector**. Dieser kann ein beliebiger Integer, **Enumeration** oder **Boolean** Typ sein. Er ist entsprechend gekennzeichnet. Von den anderen Elementen ist jeweils nur der durch den **Selector** bezeichnete Zweig aktiv.
	Datenelement vom Typ **Array**. Die Größe des **Arrays** wird im Text angezeigt.
	Datenelement vom Typ **Limited Sequence** oder **Unlimited Sequence**. Die maximale

	und aktuelle Länge wird im zugehörigen Text angezeigt.
	Datenelement vom Typ **Function Reference**.
	Startsymbol zur Kennzeichnung eines leeren **DCD**-Baum Controls. Wird jeweils angezeigt wenn keine **DCD** in den Baum geladen wurde. Es kann jedoch im Hintergrund noch eine **DCD** geladen sein [5.5].
	Icon, das verwendet wird wenn undefinierte Probleme aufgetreten sind.

Tabelle 4: DCD-Baum: Liste der Symbole

Einige Symbole können auch Schwarz/Weiß dargestellt werden. Das ist z.B. der Fall, wenn ein **Communication Object** oder eine **Operation** durch eine Ableitung überschrieben wurde und somit nicht mehr zugreifbar ist oder ein **Enumerator** oder ein Zweig einer **Union** als nicht aktiv erkannt wurde.

5.5.2 Parametrieren und Laden von Device Drivers

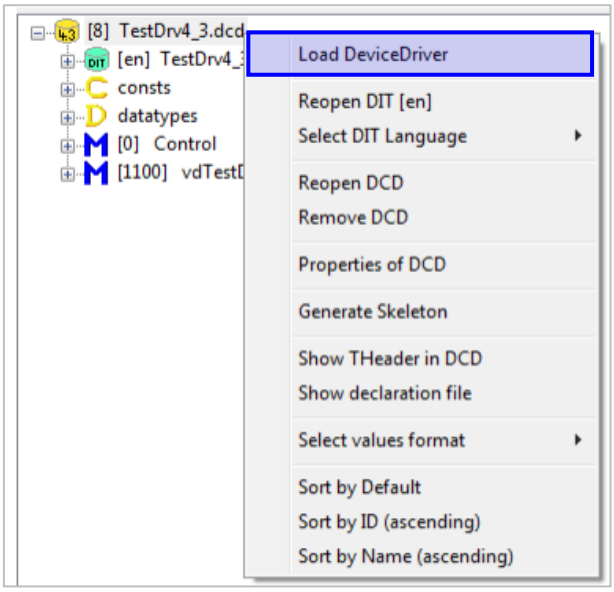


Bild 23: DCD-Knoten → Load DeviceDriver

Über den Kontextmenüpunkt `Load DeviceDriver` eines **DCD**-Knotens kann ein zur ausgewählten **DCD** passender **Device Driver** geladen werden. Hierzu erscheint ein Dialog mit den vorzunehmenden Einstellungen für den Ladevorgang des **Device Driver**.

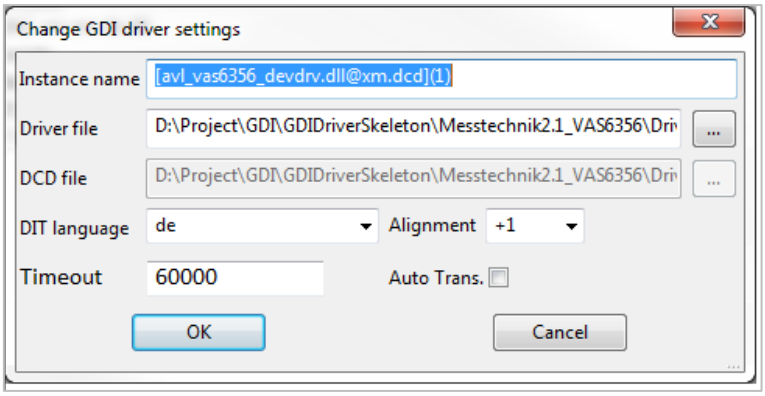


Bild 24: Parameterdialog: Laden eines Device Driver

Der Dialog zeigt die Einstellungen, welche der Benutzer vor dem Laden eines **Device Driver** als Grundlage zur fehlerfreien Arbeit mit dem **Device Driver** vorgeben muss.

5.5.2.1 Eingabe eines Instanznamens

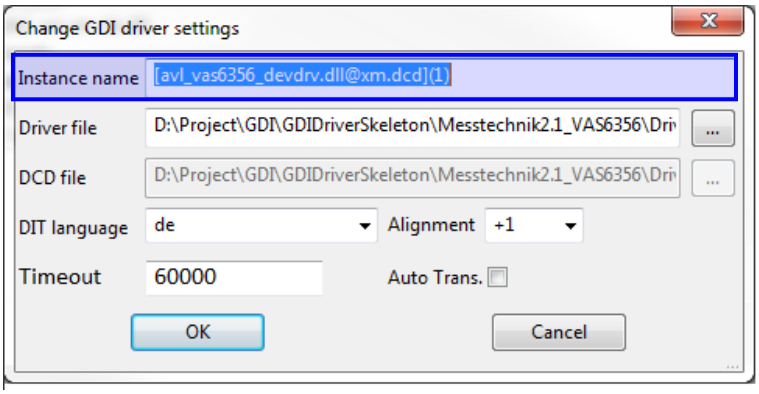


Bild 25: Device Driver – Eingabe des Instanznamens

Für jeden Ladevorgang wird eine Instanz im Instanzen-Baum [5.6] angelegt. Der hier unter `Instance name` eingetragene Name wird von **GINA2010** vorgeschlagen, kann jedoch verändert werden. Er dient als Bezug für alle auf diesem Objekt weiterhin auszuführenden Aktionen.

5.5.2.2 Auswahl des Device Driver

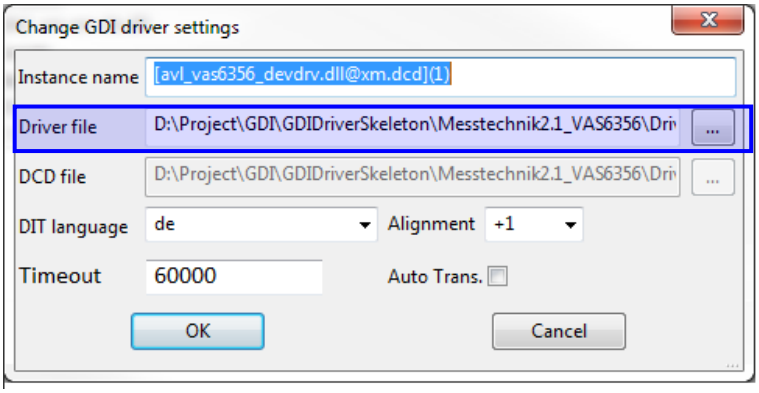


Bild 26: Device Driver – Auswahl der Treiber-Bibliothek

Unter `Driver file` kann der Dateipfad des zu ladenden **Device Driver** im Textfeld eingetragen oder mittels `...` und dem darauffolgenden `Systemdialog` zum Öffnen von Dateien ausgewählt werden.

5.5.2.3 Auswahl der DCD-Datei

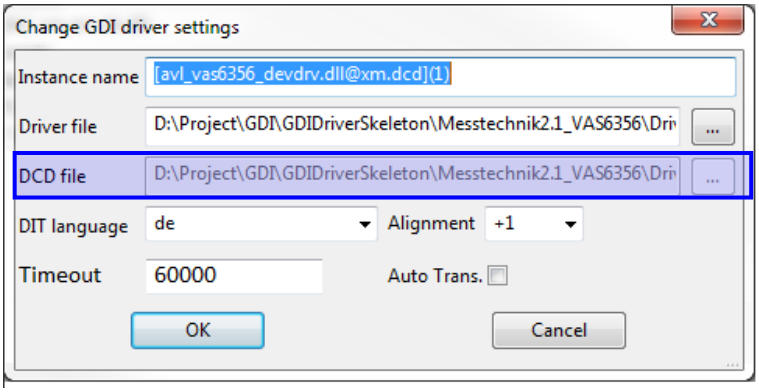


Bild 27: Device Driver – Auswahl der DCD-Datei

Unter `DCD file` kann der Dateipfad der zu ladenden **DCD** im Textfeld eingetragen oder mittels `...` und dem darauffolgenden `Systemdialog` zum Öffnen von Dateien ausgewählt werden.

5.5.2.4 Auswahl der DIT-Sprache

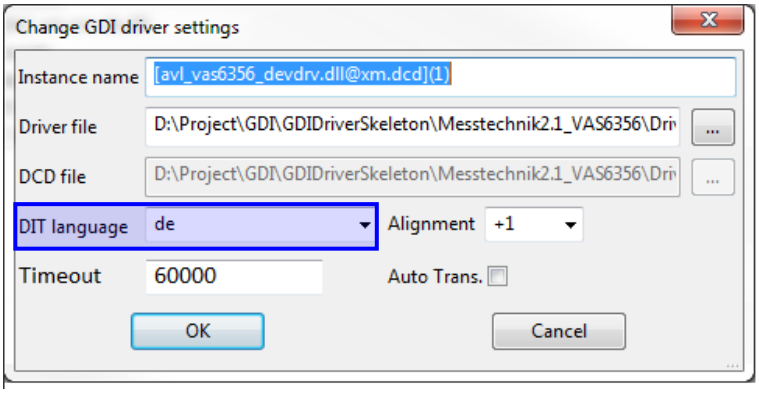


Bild 28: Device Driver – Auswahl der DIT-Sprache

Unter DIT language erlaubt eine Combobox die Auswahl einer von der **DCD** unterstützten Sprache. Fehler, Hinweise und Meldungen während der Arbeit mit dem **Device Driver** werden dann in der entsprechenden Sprache in der Ergebnisansicht [5.8] angezeigt.

5.5.2.5 Einstellen des DCD-Alignments

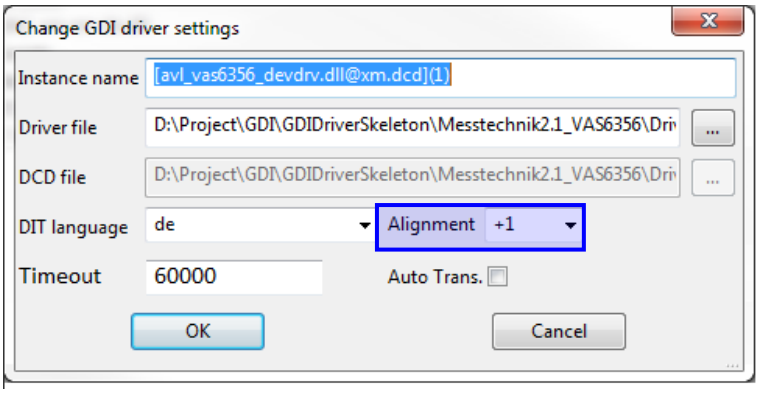


Bild 29: Device Driver – Einstellen des Alignments

Unter Alignment erlaubt eine Combobox die Auswahl des Alignments, mit welchem der **Device Driver** seine **DCD**-Strukturen [4.1.3] übersetzt hat.

Das Alignment ist beim Hersteller des **Device Driver** zu erfragen und korrekt einzusetzen.

Device Drivers ab **ASAM GDI** Version 4.5 liefern das Alignment über den API-Aufruf `GDI_Attach`, welches die hier vom Benutzer vorgenommene Vorgabe überlagert.

Die in **GINA2010** integrierte **Coordinator Engine** stellt sich auf das entsprechende Alignment des **Device Driver** als Grundvoraussetzung zur fehlerfreien Kommunikation ein.

5.5.2.6 Eingabe des Timeouts

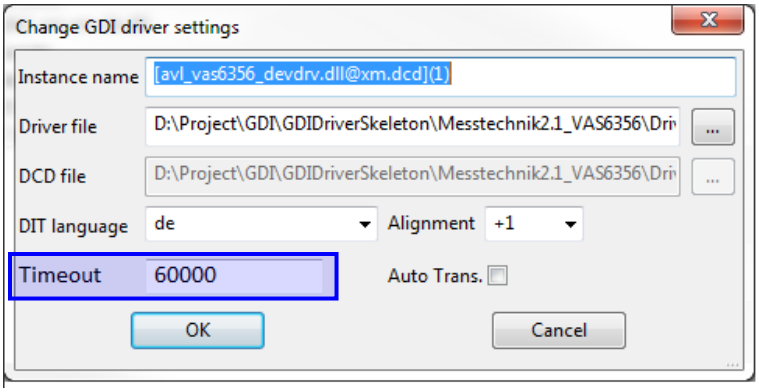


Bild 30: Device Driver – Eingabe des Timeouts

Das hier einstellbare Timeout dient zur Einstellung einer maximalen Wartezeit in Millisekunden für Aufrufe in den **Device Driver**. Damit soll bei eventuell auftretenden Blockaden im **Device Driver** die Steuerung an die Anwendung zurückgegeben werden können.

Diese Einstellung ist für Aufrufe von **Operations** gültig.

5.5.2.7 Auswahl des Betriebsmodus

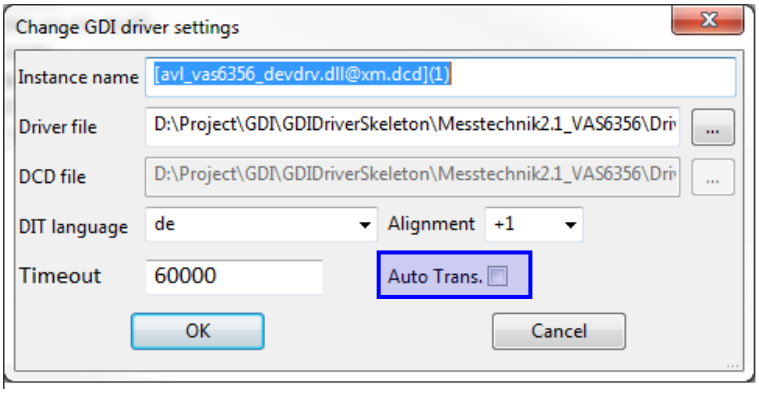


Bild 31: Device Driver – Auswahl des Betriebsmodus

Wird die als Auto Trans. bezeichnete Checkbox gesetzt (default), so führt die **Coordinator Engine** alle nötigen **GDI Phasen**-Übergänge automatisch aus. Die entsprechende **GDI Phase** wird von **GINA2010** nach jeder relevanten Aktion abgefragt und im Eintrag des entsprechenden **Virtual Device** im Instanzen-Baum [5.6] angezeigt. Dieses Verhalten wird im Folgenden als Automatischer **Betriebsmodus** bezeichnet.

Ist diese Checkbox nicht gesetzt, so muss das **Control VD** manuell angelegt werden. Dadurch wird es ebenfalls im Instanzen-Baum dargestellt. Die **Operations** von **Transition Function** sind nun manuell aufzurufen, um die nötigen **GDI Phasen** zu erreichen. Dieses Verhalten wird im Folgenden als Manueller **Betriebsmodus** bezeichnet [7.3.2].

5.5.2.8 Laden des Device Driver

Nach Bestätigung dieses Dialogs mittels Schalter OK wird der **Device Driver** geladen und zur Präsentation dieses ein neuer Root-Knoten in den Instanzen-Baum [5.6] eingefügt. Diesem Knoten sollte das blaue Symbol zugeordnet sein, anderenfalls liegt ein Problem vor.

Fehler und andere Ausgaben, welche beim Laden eines **Device Driver** entstehen, werden in der Log-Ansicht [5.9] sichtbar.

Das Entladen eines **Device Driver** wird im Kontextmenü des **Device Driver**-Knotens im Instanzen-Baum [5.6] über den Menüpunkt Close DeviceDriver ermöglicht.

Weitere Aktionen werden über das Kontextmenü des Instanzen-Baum ermöglicht.

5.5.3 Aktualisieren des DCD-Baums und der DIT-Sprache

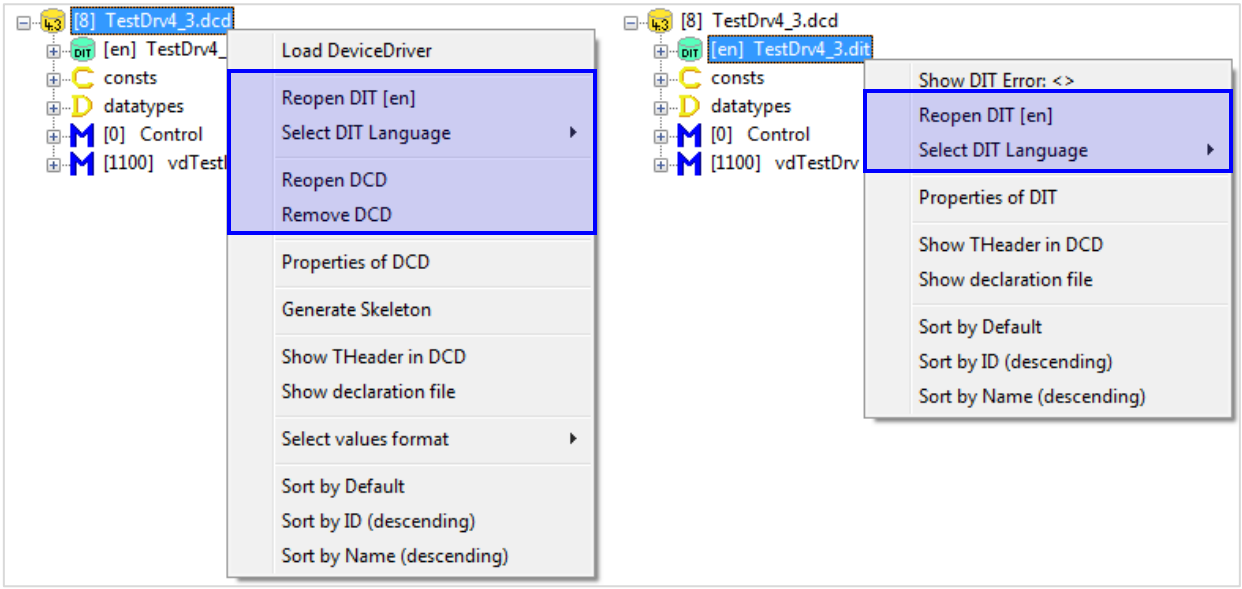


Bild 32: Aktualisieren des DCD-Baums

- **Reopen DIT**

  Durch Anwahl dieses Menüpunktes ist es möglich, die zur **DCD** geladene **DIT** nochmals einzulesen. Dies kann nach vorgenommenen Änderungen außerhalb **GINA2010** hilfreich sein, um die Inhalte zu aktualisieren.

- **Select DIT Language**

  Durch Anwahl dieses Menüpunktes kann in eine andere durch die **DCD** unterstützte Sprache gewechselt werden. Die hier eingestellte Sprache ist nur für die Anzeige der Fehlertexte im **DCD**-Baum gültig. Jeder **Device Driver** wird entsprechend Kapitel 5.5.2.4 zur Ladezeit auf eine eigene Sprache eingestellt.

- **Reopen DCD**

  Durch Anwahl dieses Menüpunktes ist es möglich, die geladene **DCD** inkl. **DIT** nochmals einzulesen. Dies kann nach vorgenommenen Änderungen außerhalb **GINA2010** hilfreich sein, um die Inhalte zu aktualisieren.

- Remove DCD

  Durch Anwahl dieses Menüpunktes ist es möglich, die geladene **DCD** zu entladen und aus dem **DCD**-Baum zu entfernen.

5.5.4 Allgemeine DCD-Informationen

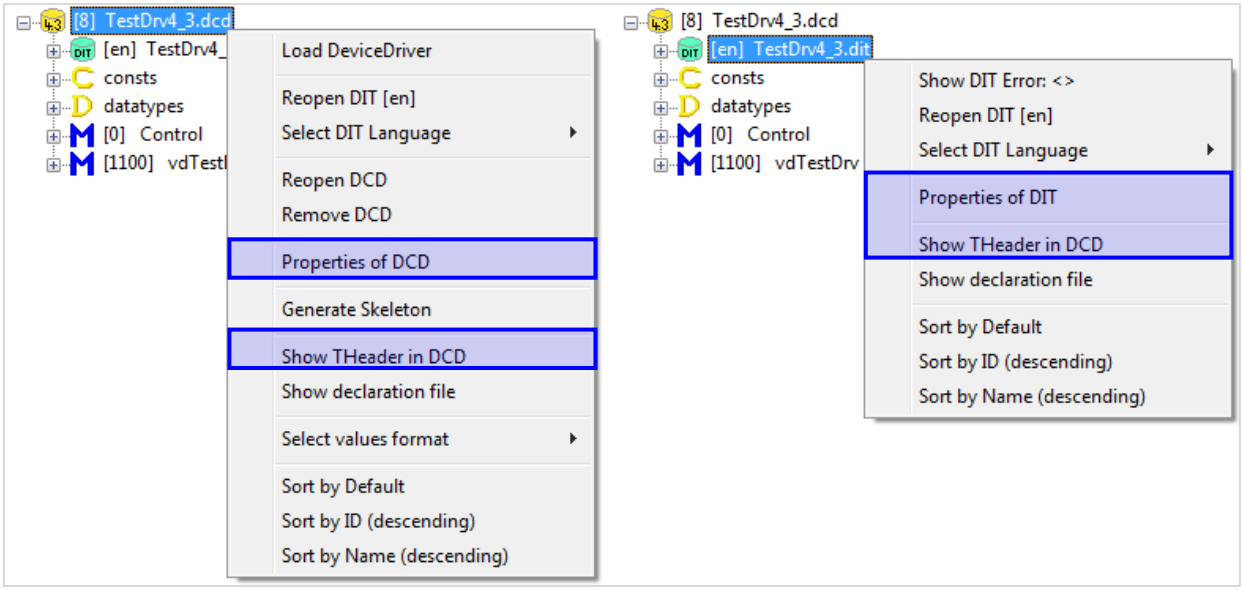


Bild 33: Allgemeine DCD-Informationen

- Properties of DCD

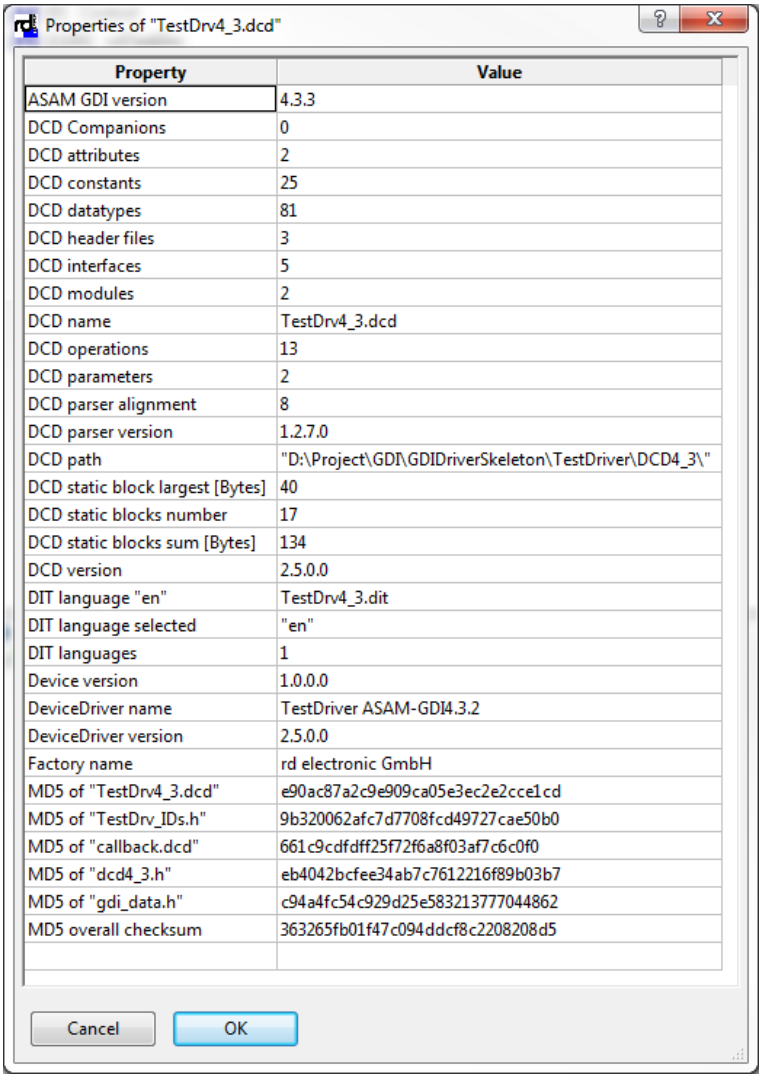


Bild 34: DCD-Properties

Der Dialog zeigt einige Informationen nach dem Laden einer **DCD**, welche für spätere Fehlersuchen hilfreich sein können.

- Show Properties of DIT

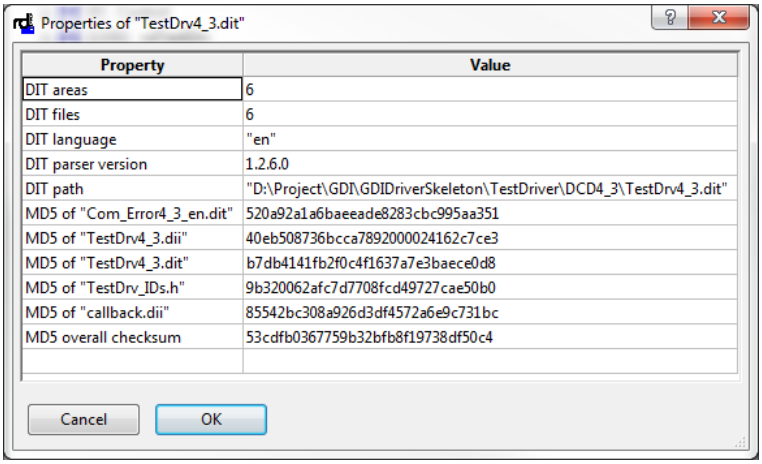


Bild 35: DIT-Properties

Der Dialog zeigt einige Informationen nach dem Laden einer **DIT**, welche für spätere Fehlersuchen hilfreich sein können.

- Show THeader in DCD

Dieser Menüpunkt öffnet mit dem eingestellten externen Texteditor [5.3.4] die **DCD**-Datei und setzt den Cursor an die Zeile, in welcher die THeader-Struktur [GDI-Doc3] der **DCD** definiert ist.

5.5.5 Öffnen des Skeleton Generator

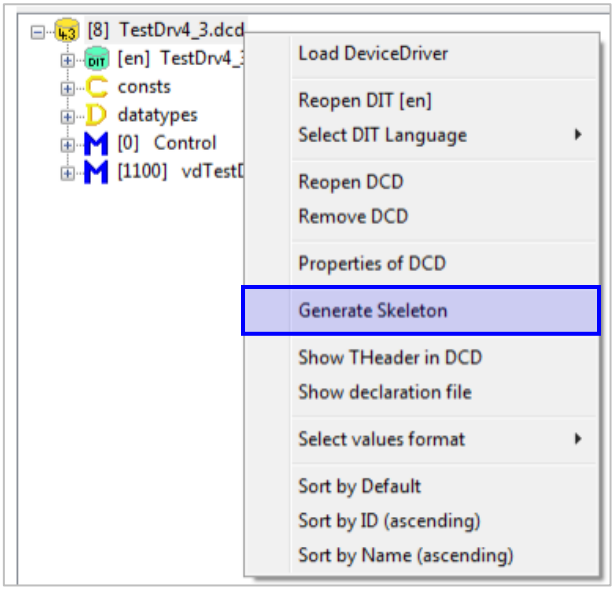


Bild 36: Öffnen des Skeleton Generator

Bei Anwahl des Menüpunktes Generate Skeleton öffnet sich der Dialog des integrierten **Skeleton Generators**, zur Generierung von **Device Drivers** entsprechend der ausgewählten **DCD**.

Für die Benutzung des Skeleton Generator ist der Kauf einer entsprechenden Lizenz [2.2] erforderlich.

Die Benutzung des Skeleton Generator wird in Kapitel 6 beschrieben.

5.6 Instanzen-Baum

Links unten befindet sich der Instanzen-Baum. Nach dem Programmstart ist dieser Bereich leer. Hier erscheinen später die angelegten Instanzen der **Device Driver**. Wurzel der angezeigten Bäume ist jeweils der geladene **Device Driver**.

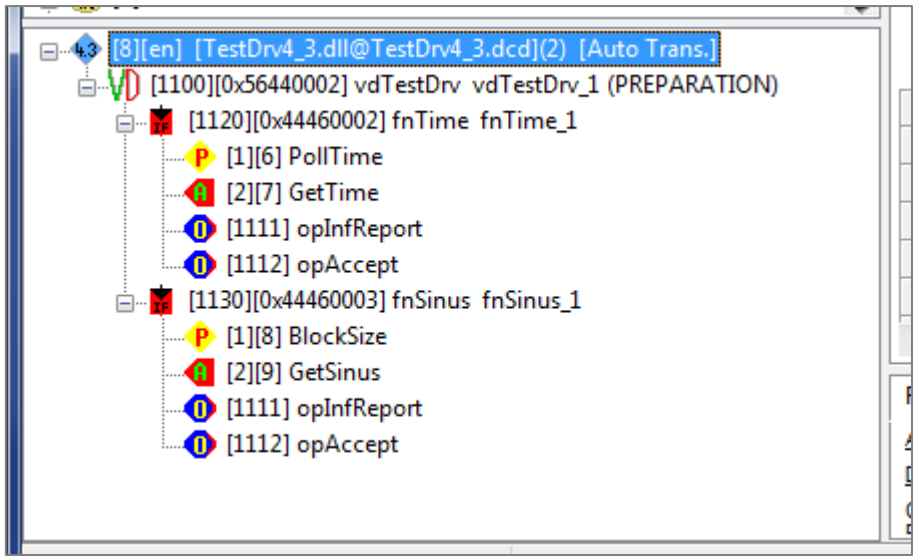


Bild 37: Instanzen-Baum

Im DCD-Baum stehen ständig die Allgemeinen Funktionen entsprechend Kapitel 5.2 zur Verfügung.

5.6.1 Liste der Symbole

Im Folgenden werden die verwendeten Symbole, die zur Darstellung der Instanzen dienen, kurz beschrieben.

	Ein **Device Driver** nach **GDI** 4.2 oder 4.3 wurde erfolgreich geladen.
	Ein **Device Driver** nach **GDI** 4.2 oder 4.3 wurde geladen, es sind jedoch Probleme beim Laden aufgetreten. Am wahrscheinlichsten war die Version der **DCD** nicht identisch mit der Versionsnummer des **Device Driver**. Dies wird auch durch eine entsprechende Messagebox angezeigt.
	Eine Instanz vom **DCD**-Typ **Control VD** wurde angelegt. Dies erzwingt den Manuellen **Betriebsmodus** [5.5.2.7]. Das **Control VD** hat eine Sonderstellung und wird von der **Coordinator Engine** auch speziell behandelt. Deshalb hat es ein eigenes Symbol. Die weitergehende Benutzung ist identisch mit anderen **Virtual Devices**.
	**Virtual Device** Eine Instanz vom **DCD**-Typ **Module** wurde angelegt.
	**Function Object** Eine Instanz vom **DCD**-Typ **Interface** wurde angelegt.
	**Parameter** (**Communication Object**) Eine Instanz eines readonly oder read/write **Parameter** wurde angelegt.
	**Attribute** (**Communication Object**) Eine Instanz eines readonly oder read/write **Attribute** wurde angelegt.
	**Operation** Eine Instanz vom **DCD**-Typ **Operation** wurde angelegt. Ein roter Pfeil kennzeichnet dabei mögliche Eingabeparameter, ein grüner Pfeil mögliche Ausgabeparameter.

Tabelle 5: Instanzen-Baum: Liste der Symbole

Zusätzlich können auch die Symbole des **DCD**-Baums [5.5] in diesem Baum dargestellt werden. Dies kann z.B. vorkommen wenn durch Drag und Drop ein Teil eines **DCD**-Baums nach hier kopiert wurde. Eine derartige Kopie kann jedoch in diesem Baum nur dargestellt werden. Es ist keine Benutzung (z.B. Aufruf von „Initiate“) möglich.

5.6.2 Arbeit mit geladenen Device Drivers

Die weitere Arbeit mit geladenen **Device Drivers** wird über den Instanzen-Baum ermöglicht. Jede Instanz bietet entsprechend ihres Typs ein Kontextmenü mit mitunter möglichen Aufrufen der **Device Drivers** an.

Die Funktionen der Kontextmenüs der hier dargestellten Instanzen sind anhand eines Ablaufbeispiels im Kapitel 7 erläutert.

5.7 Schrittliste

Im rechten Quadranten befindet sich die Schrittliste. In ihr werden alle Aktionen mit einem **Device Driver** aufgezeichnet. Diese Liste kann als **Makro**-Datei gespeichert und später erneut geladen und ausgeführt werden.

Das folgende Bild zeigt die Liste nach der Erzeugung der Instanzen im oben abgebildeten Instanzen-Baum [5.6].

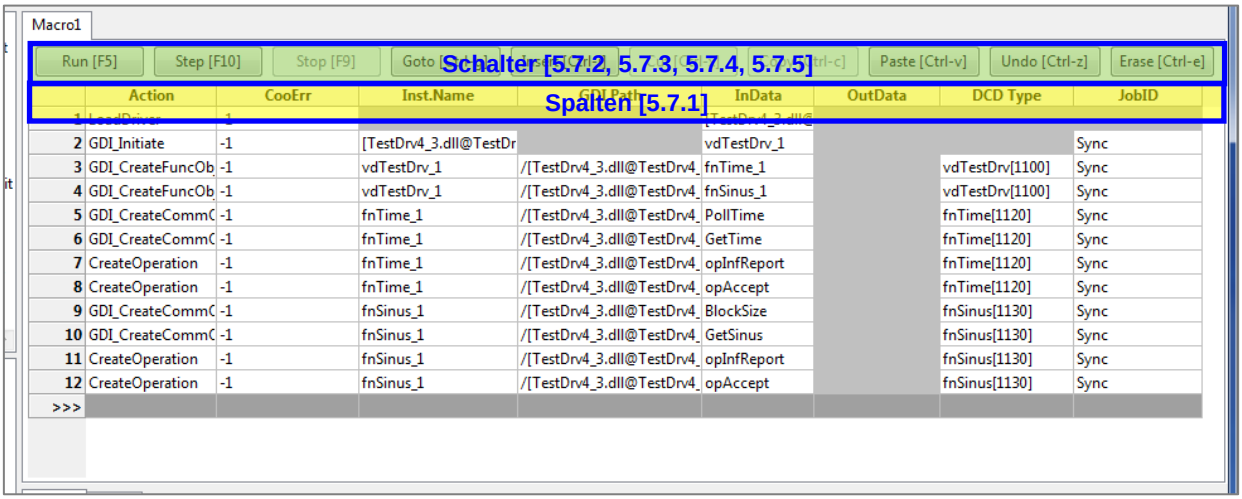


Bild 38: Schrittliste

5.7.1 Spalten

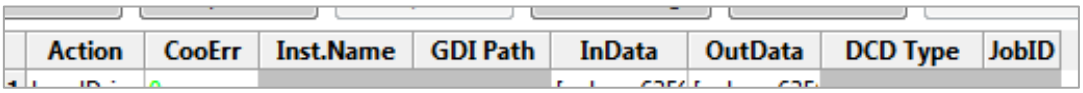


Bild 39: Schrittliste: Spalten

Die einzelnen Spalten sind im Folgenden kurz erläutert.

- Action

  Diese Spalte enthält eine kurze Bezeichnung der ausgeführten Aktion. Das können Programminterne bzw. **Coordinator Engine** Aktionen sein (z.B. `CreateOperation`) oder Aktionen, die im Wesentlichen einen bestimmten Funktionsaufruf auf der **Device Driver** API ausführen. Letztere beginnen alle mit „`GDI_`“. Es ist zu beachten, dass in Abhängigkeit vom **Betriebsmodus** [5.5.2.6] mehrere Funktionsaufrufe an der API erfolgen können (z.B. zusätzlich `GDI_Status` um die aktuelle **GDI Phase** zu ermitteln). Dargestellt wird aber nur der funktional beabsichtigte Aufruf.

- CooErr

  Dies ist der Returncode der in **GINA2010** verwendeten **Coordinator Engine**.

  - Der Wert `-1` wird angezeigt, wenn der ausgewählte Schritt noch nicht ausgeführt wurde.
  - Der Wert `0` wird angezeigt, wenn der ausgewählte Schritt fehlerfrei ausgeführt wurde.
  - Der Wert `1` wird angezeigt, wenn der ausgewählte Schritt asynchron ausgeführt wurde oder noch in Ausführung ist. Asynchrone Aufrufe werden jedoch derzeit von **GINA2010** nicht unterstützt.
  - Alle anderen Werte werden rot dargestellt und deuten auf Fehler bei der Ausführung des ausgewählten Schrittes hin.

    Diese stammen entweder aus der Verarbeitung des Kommandos selbst oder der **Coordinator Engine**. Im Ergebnisansicht [5.8] wird jeweils eine kurze Beschreibung angezeigt. Zwei Werte sollten gesondert erwähnt werden. Ein Wert von `-1008` bedeutet einen Fehler des **Device Driver** und ein Wert von `-1004` einen Timeout im **Coordinator Engine**. Dass heißt, der Aufruf zum **Device Driver** ist nach einer bestimmten Zeit nicht zurückgekehrt und kehrt möglicherweise nie mehr zurück (**Device Driver** klemmt).

- Inst.Name

  Name der Instanz, auf der die Aktion ausgeführt wird. Z.B. wird `GDI_CreateCommObject` auf der Instanz eines **Function Object** ausgeführt. Bei erzeugenden Aktionen ist dies immer der Name des hierarchisch über dem zu erzeugenden Objekt liegenden Objekts. Folglich hat `LoadDriver` keinen Eintrag. Bei zugreifenden Aktionen (z.B. `GDI_Read`) ist dies der Name des benutzten Objekts.

- GDI Path

  Diese Spalte beinhaltet einen virtuellen Pfad dessen erster Eintrag der **Device Driver** ist, dargestellt durch die numerische ID des **Device Driver**. Diese ID ist ein von **GINA2010** vergebener Wert. Danach folgen alle Instanz-Namen der Objekt Hierarchie. Der Inst.Name gehört nicht mit zum Pfad. Um das exakte Objekt zu ermitteln ist der GDI Path im Zusammenhang mit dem Inst.Name zu sehen. Bei kleineren Projekten kann dadurch die Spalte GDI Path minimiert werden ohne die Übersicht zu verlieren.

- InData

  Diese Spalte enthält die Eingabeparameter für eine bestimmte Aktion. Dies ist möglicherweise nur der gewünschte Instanz-Name der direkt in diesem Feld editiert werden kann. Bei komplexeren Aktionen, wie z.B. dem Schreiben eines **Parameter**, öffnet ein Doppelklick einen Eingabedialog zum Editieren der gewünschten Eingabewerte [5.10].

- OutData

  Diese Spalte enthält Ergebniswerte nachdem die Aktion bereits ausgeführt wurde und solche Werte vorliegen. Auch dies ist möglicherweise nur der Instanz-Name, der beim Anlegen des Objektes angegebenen wurde. Zur Vermeidung von Nameskonflikten kann es hier zu Abweichungen kommen. Diese werden hier sichtbar. Bei komplexeren Aktion wie z.B. dem Lesen eines **Parameter**, kann durch einen Doppelklick das Ergebnis in einem Dialog betrachtet werden [5.10].

- DCD Type

  Hier wird wenn verfügbar der Typname und die ID aus der **DCD** zu dem Objekt angegeben, dessen Instanz-Name in Inst.Name eingetragen ist.

- JobID

  Diese Spalte ist für die Darstellung der bei asynchroner Kommunikation verwendeten Job ID vorgesehen. Zur Zeit ist diese jedoch an der Oberfläche noch nicht verfügbar, so dass hier immer 0 angezeigt wird.

Alle Zeilen dieser Liste sind eigenständig. Jede Zeile enthält alle nötigen Informationen um die benötigten Objekte zu finden. Dies erfolgt anhand der vergebenen Instanz-Namen.

Es ist zu beachten, dass einige Spalten nach dem Laden eines **Makros** noch nicht ausgefüllt sind. Das betrifft auch die Spalte DCD Type. Dies ist darin begründet, dass die Objekte anhand ihres Instanz-Namens identifiziert werden und die Information zu den Typen und IDs in der **DCD** nicht im **Makro** verfügbar ist. Die Identifizierung ist jedoch nicht möglich, solange die Objekte noch nicht angelegt wurden. Diese Spalten werden nach dem Ausführen der entsprechenden Zeile aktualisiert.

5.7.2 Steuerung der Abarbeitung

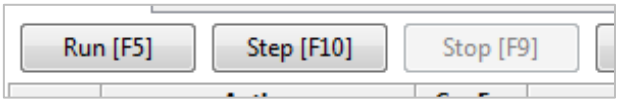


Bild 40: Schrittliste: Steuerung der Abarbeitung

5.7.2.1 Run

Bei Anwahl dieses Schalters oder durch Betätigen der Funktionstaste *F5* startet die Abarbeitung der angezeigten Schritte ab der aktuell markierten Zeile bis zur letzten Zeile der Schrittliste.

5.7.2.2 Step

Bei Anwahl dieses Schalters oder durch Betätigen der Funktionstaste *F10* wird nur die markierte Zeile abgearbeitet.

5.7.2.3 Stop

Befindet sich die Schrittliste in Abarbeitung, so kann diese durch Anwahl dieses Schalters oder durch Betätigen der Funktionstaste *F9* unterbrochen werden.

Eine Unterbrechung ist jedoch nur zwischen den Aufrufen in den **Device Driver** möglich. Aktive Aufrufe können damit nicht unterbrochen werden.

5.7.3 Navigation in der Schrittliste

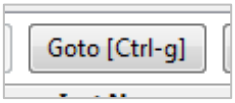


Bild 41: Schrittliste: Navigation in der Schrittliste

5.7.3.1 Goto

Dieser Schalter dient zur schnelleren Navigation in der Schrittliste. Bei Anwahl oder durch Verwendung des Tastenkürzels *Ctrl-g* erscheint ein Auswahldialog, in den die gewünschte Zeilennummer eingegeben werden kann.

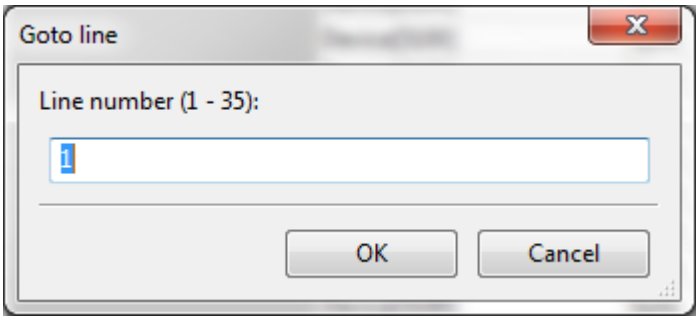


Bild 42: Schrittliste: Zeilennavigation „Goto“

Durch Bestätigen des Dialogs mittels OK wird der Dialog geschlossen und die angegebene Zeilennummer in der Schrittliste selektiert.

Durch Abbrechen des Dialogs mittels Cancel wird der Dialog geschlossen. Die aktuell selektierte Zeile ändert sich dabei nicht.

5.7.4 Einfügen von Kontrollelemente

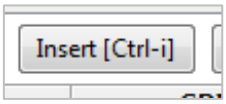


Bild 43: Schrittliste: Einfügen von Kontrollelementen

5.7.4.1 Insert

Durch Anwahl dieses Schalters oder durch Verwendung des Tastenkürzels *Ctrl-i* erscheint ein Dialog zur Auswahl der einzufügenden Kontrollelementes.

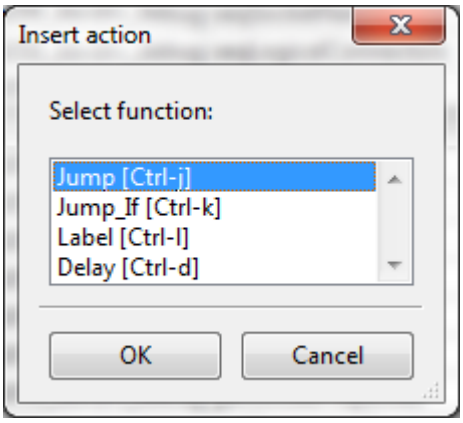


Bild 44: Schrittliste: Kontrollelemente

Die verfügbaren Kontrollelemente sind in den folgenden Kapitel beschrieben.

5.7.4.2 Insert - Jump

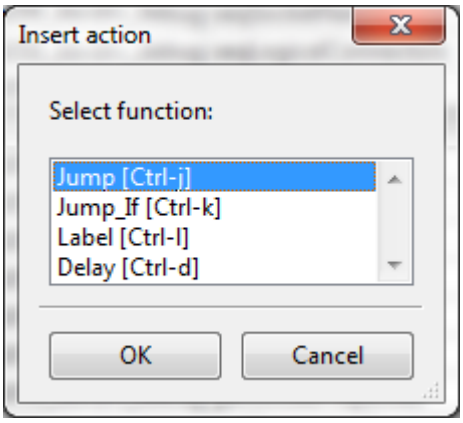


Bild 45: Schrittliste: Auswahl Kontrollelement Jump

Durch Auswahl von Jump und Bestätigen durch Anwahl des Schalters OK wird in die Schrittliste eine Zeile eingefügt, welche in der Spalte Action [5.7.1] den Inhalt „Jump“ anzeigt.

Das Einfügen eines Jump kann auch durch Verwendung des Tastenkürzels *Ctrl-j* erreicht werden.

Durch Einfügen eines Jump wird ein unbedingter Sprung in den Ablauf eingefügt. In der Spalte Inst.Name [5.7.1] ist das Sprungziel anzugeben.

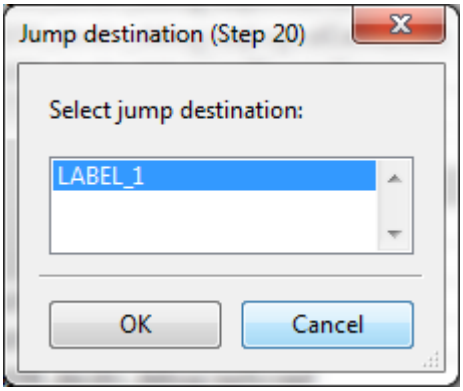


Bild 46: Schrittliste: Sprungziel

Dazu muss der Name eines vorhandenen Label ausgewählt werden, ansonsten wird das Sprungziel als ungültig („<INVALID>“) markiert.

In der Spalte InParam [5.7.1] kann die Anzahl der Sprungwiederholungen angegeben werden.

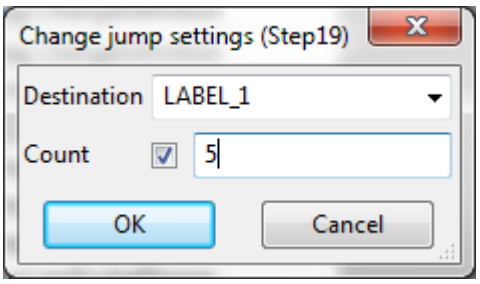


Bild 47: Schrittliste: Parameter unbedingter Sprünge

In der Spalte OutParam [5.7.1] erscheint dann die aktuelle Anzahl von Durchläufen zusammen mit der Anzahl der Sprungwiederholungen, die bei jedem Durchlauf aktualisiert wird.

Wird keine Begrenzung der Anzahl der Sprünge benötigt, kann in InParam der Wert 0 bzw. „INFINITE“ angegeben werden. Die Spalte OutParam wird in diesem Fall nicht verwendet.

Das Kontrollelement Jump springt zum angegebenen Sprungziel (Label) innerhalb des Makro-Ablaufs und setzt den Ablauf am angegebenen Sprungziel fort. Die Anzahl der Sprungwiederholungen bestimmt die Häufigkeit der Sprungausführungen. Erreicht die aktuelle Anzahl von tatsächlich ausgeführten Sprüngen die Anzahl der parametrierten Sprungwiederholungen, so wird nicht gesprungen sondern aktuelle Anzahl von Durchläufen wieder auf 0 und der Rückgabewert auf -1 gesetzt. In diesem Fall wird mit dem nachfolgenden Makro-Schritt fortgesetzt.

Wurde keine Anzahl von Sprungwiederholungen parametriert, wird das angegebene Sprungziel immer angesprungen. In der Spalte CooErr [5.7.1] wird mit dem Rückgabewert 0 das Ausführen des Sprunges angezeigt, entsprechend zeigen der Wert -1 die Nichtausführung und der Wert -32756 ein ungültiges Sprungziel an. Der Rückgabewert hat keinen Einfluss auf die Auswertung von Sprungbedingungen.

5.7.4.3 Insert - Jump_If

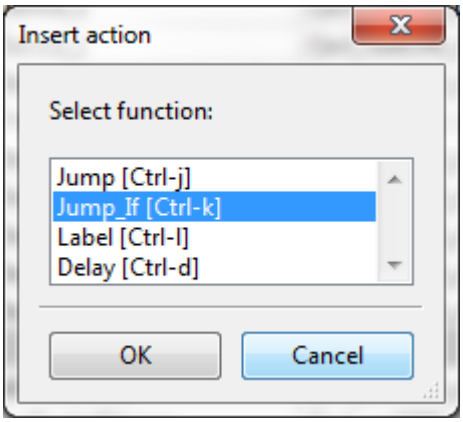


Bild 48: Schrittliste: Auswahl Kontrollelement Jump_If

Durch Auswahl von Jump_If und Bestätigen durch Anwahl des Schalters OK wird in die Schrittliste eine Zeile eingefügt, welche in der Spalte Action [5.7.1] den Inhalt „Jump_If“ anzeigt.

Das Einfügen eines Jump_If kann auch durch Verwendung des Tastenkürzels *Ctlr-k* erreicht werden.

Durch Einfügen eines Jump_If wird ein bedingter Sprung in den Ablauf eingefügt. In der Spalte Inst.Name ist das Sprungziel anzugeben [Bild 46]. Dazu muss der Name eines vorhandenen Label ausgewählt werden, ansonsten wird das Sprungziel als ungültig markiert. In der Spalte InParam kann die Anzahl der Sprungwiederholungen zusammen mit der Bedingung für das Ausführen des Sprungs angegeben werden. In der Spalte OutParam erscheint die aktuelle Anzahl von Durchläufen zusammen mit der Anzahl der Sprungwiederholungen, die bei jedem Durchlauf aktualisiert wird.

Wird keine Begrenzung der Anzahl der Sprünge benötigt, kann der zugehörige Wert weggelassen werden. Die Bedingung für das Ausführen des Sprungs bezieht sich immer auf das Ergebnis der letzten Aktion, die keins der Kontrollelemente Jump, Jump_If, Label und Delay ist. Dieses Ergebnis beinhaltet den **Coordinator Engine**-Fehlercode CooErr sowie die **Device Driver** Ergebnisstruktur bestehend aus den Elementen RC, Qual, Grade und Code. Dementsprechend besteht die Bedingung für das Ausführen des Sprungs aus fünf optionalen Termen, die sich auf diese Werte beziehen.

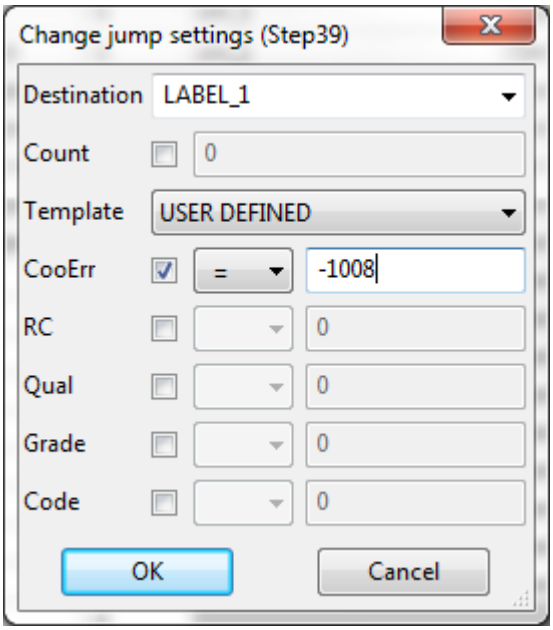


Bild 49: Schrittliste: Parameter bedingter Sprünge

Die Terme sind miteinander UND-verknüpft und enthalten jeweils einen Vergleichsoperator und einen dazugehörigen Vergleichswert.

Folgende Vergleichsoperatoren stehen zur Verfügung:

Operator	**Beschreibung**
=	Die Teilbedingung ist erfüllt, wenn der Vergleichswert **gleich** dem zugeordneten Wert des letzten Aktionsergebnisses ist
<	Die Teilbedingung ist erfüllt, wenn der Vergleichswert **kleiner als** der zugeordnete Wert des letzten Aktionsergebnisses ist
>	Die Teilbedingung ist erfüllt, wenn der Vergleichswert **größer als** der zugeordneten Wert des letzten Aktionsergebnisses ist
<>	Die Teilbedingung ist erfüllt, wenn der Vergleichswert **ungleich** dem zugeordneten Wert des letzten Aktionsergebnisses ist
<=	Die Teilbedingung ist erfüllt, wenn der Vergleichswert **kleiner oder gleich** dem zugeordneten Wert des letzten Aktionsergebnisses ist
>=	Die Teilbedingung ist erfüllt, wenn der Vergleichswert **größer oder gleich** dem zugeordneten Wert des letzten Aktionsergebnisses ist

Tabelle 6: Schrittliste: Operatoren für bedingte Sprünge

Jeder Term kann einzeln aktiviert werden, um nur bestimmte Ergebniswerte zu berücksichtigen und andere Ergebniswerte ignorieren zu können. Falls ein gültiges letztes Aktionsergebnis gesetzt jedoch keiner der fünf optionalen Terme aktiviert wurde, verhält sich das Kontrollelement `Jump_If` genauso wie das Kontrollelement `Jump`.

Das Kontrollelement `Jump_If` springt bei erfüllter Bedingung und gültigem letztem Aktionsergebnis zum angegebenen Sprungziel innerhalb des **Makro**-Ablaufs und setzt den Ablauf am angegebenen Sprungziel fort. Die Anzahl der Sprungwiederholungen bestimmt die Häufigkeit der Sprungausführungen. Erreicht die aktuelle Anzahl von tatsächlich ausgeführten Sprüngen die Anzahl der parametrierten Sprungwiederholungen, so wird nicht gesprungen sondern aktuelle Anzahl von Durchläufen wieder auf `0` und der Rückgabewert auf `-1` gesetzt. In diesem Fall oder auch wenn die parametrierte Bedingung nicht erfüllt ist, wird mit dem nachfolgenden **Makro**-Schritt fortgesetzt. Wurde keine Anzahl von Sprungwiederholungen parametriert, wird das angegebene Sprungziel bei erfüllter Bedingung immer angesprungen. In der Spalte `CooErr` wird mit dem Rückgabewert `0` das Ausführen des Sprunges angezeigt, entsprechend zeigen der Wert `-1` die Nichtausführung und der Wert `-32756` ein ungültiges Sprungziel an. Der Rückgabewert hat keinen Einfluss auf die Auswertung von Sprungbedingungen.

Die Spalte `InParam` zeigt eine Übersicht über die vorgenommene Parametrierung eines `Jump_If`.

5.7.4.4 `Insert` - `Label`

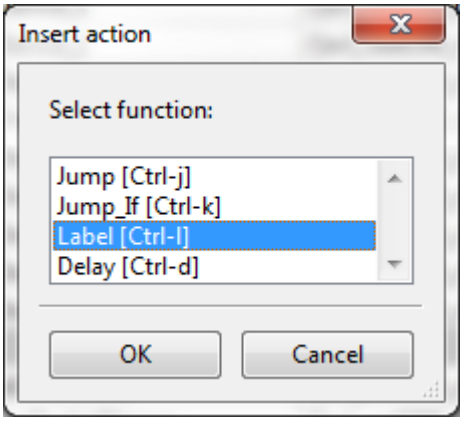


Bild 50: Schrittliste:Auswahl Kontrollelement Label

Durch Auswahl von Label und Bestätigen durch Anwahl des Schalters OK wird in die Schrittliste eine Zeile eingefügt, welche in der Spalte Action [5.7.1] den Inhalt „Label“ anzeigt.

Das Einfügen eines Label kann auch durch Verwendung des Tastenkürzels *Ctlr-l* erreicht werden.

Durch Einfügen eines Label wird eine Markierung in den Ablauf eingefügt, welches als Sprungziel für Jump und Jump-If verwendet werden kann. In der Spalte Inst.Name [5.7.1] ist ein eindeutiger Name für das Label anzugeben.

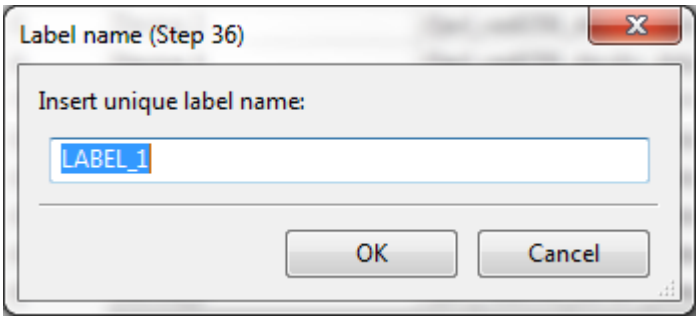


Bild 51: Schrittliste: Labelnamen

Beim Durchlaufen eines Label wird keine Aktion ausgeführt. In der Spalte CooErr wird mit dem Rückgabewert 0 das Durchlaufen des Schrittes angezeigt. Entsprechend zeigt der Wert -1 an, dass der Schritt bisher noch nicht durchlaufen wurde. Der Rückgabewert hat keinen Einfluss auf die Auswertung von Sprungbedingungen.

5.7.4.5 Insert - Delay

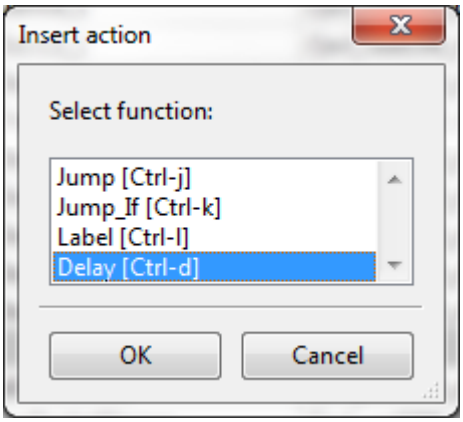


Bild 52: Schrittliste: Auswahl Kontrollelement Delay

Durch Auswahl von Delay und Bestätigen durch Anwahl des Schalters OK wird in die Schrittliste eine Zeile eingefügt, welche in der Spalte Action [5.7.1] den Inhalt „Delay“ anzeigt.

Das Einfügen eines Delay kann auch durch Verwendung des Tastenkürzels *Ctlr-d* erreicht werden.

Durch Einfügen eines Delay wird die Abarbeitung der Schrittliste um die in der Spalte InData angegebene Wartezeit in Millisekunden verzögert. Der Wert „INFINITE“ steht dabei für ein unendlich lange Wartezeit und erfordert die manuelle Aufforderung zur weiteren Abarbeitung durch den Benutzer.

5.7.5 Bearbeitung der Schrittliste

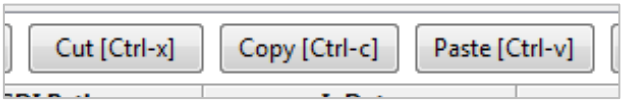


Bild 53: Schrittliste: Bearbeitung der Schrittliste

5.7.5.1 Cut

Ausschneiden (entfernen und zusammenrutschen) der markierten Zeile. Diese befindet sich dann in einem Zwischenpuffer und kann an anderer Position wieder eingefügt werden. Es ist zu beachten, dass dies nicht die Zwischenablage des Betriebssystems ist und auch nicht Programmübergreifend möglich ist.

Ein Cut kann auch durch Verwendung des Tastenkürzels Ctlr-x erreicht werden.

5.7.5.2 Copy

Kopieren der markierten Zeile in den Zwischenpuffer. Es ist zu beachten, dass dies nicht die Zwischenablage des Betriebssystems ist und auch nicht Programmübergreifend möglich ist.

Ein Copy kann auch durch Verwendung des Tastenkürzels Ctlr-c erreicht werden.

5.7.5.3 Paste

Einfügen der Zeile aus dem Zwischenpuffer an die aktuelle Position. Die nachfolgenden Zeilen werden verschoben. Es ist zu beachten, dass dies nicht die Zwischenablage des Betriebssystems ist und auch nicht Programmübergreifend möglich ist.

Ein Paste kann auch durch Verwendung des Tastenkürzels Ctlr-v erreicht werden.

5.7.5.4 Undo

Bei Anwahl dieses Schalters wird die letzte Aktion rückgängig gemacht. Es kann nur genau ein Schritt rückgängig gemacht werden.

Ein Undo kann auch durch Verwendung des Tastenkürzels Ctlr-z erreicht werden.

5.7.5.5 Erase

Löscht die gesamte Schrittliste.

Ein Erase kann auch durch Verwendung des Tastenkürzels Ctlr-e erreicht werden.

5.8 Ergebnisansicht

Im rechten unteren Quadranten unter der Karteikarte Res werden alle relevanten Informationen der letzten ausgeführten Aktion oder der zuletzt angeklickten Zeile der Schrittliste [5.7] dargestellt.

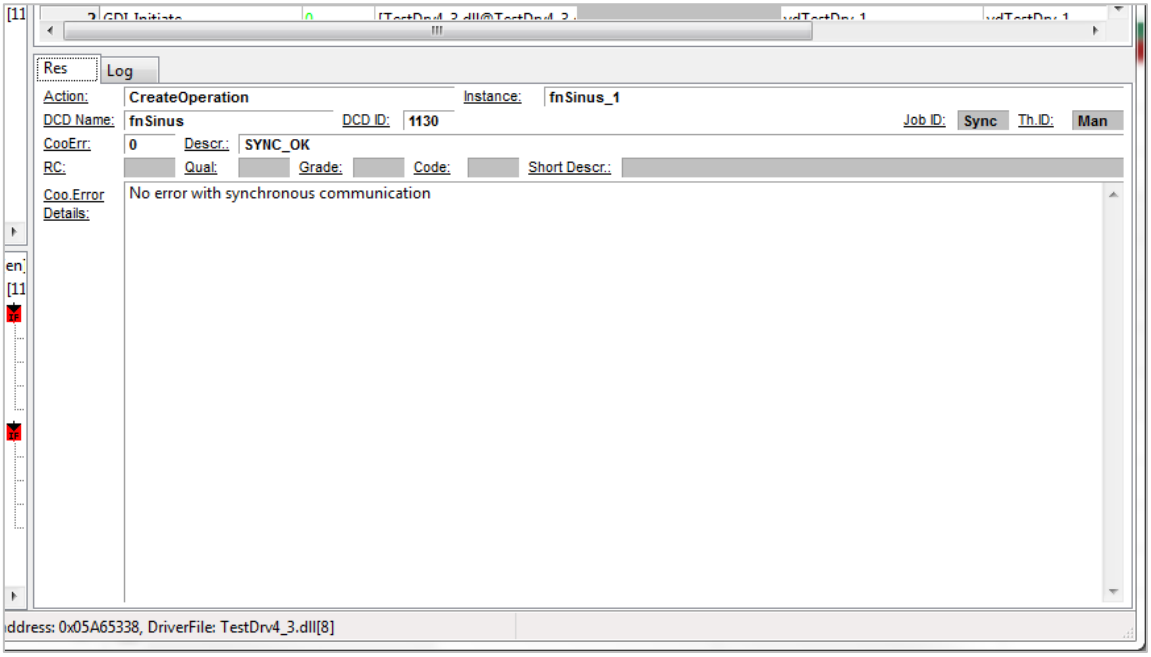


Bild 54: Ergebnisansicht

Es werden folgende Informationen angezeigt:

- Action

  Bezeichnung der ausgeführten Aktion [5.7.1].

- Instance

  Instanz-Name des Objekts auf dem die Aktion ausgeführt wurde [5.7.1].

- DCD Name

  Typname aus der DCD des in Instance angeführten Objekts [5.7.1].

- DCD ID

  Typ ID aus der **DCD** des in Instance angeführten Objekts [5.7.1].

- CooErr

  Returncode der integrierten **Coordinator Engine** oder von **GINA2010** selbst [5.7.1].

- Descr.

  Textuelle Kurzbeschreibung des in CooErr angeführten Returncodes.

- Die Folgenden Felder sind nur gefüllt, wenn ein Fehler im **Device Driver** aufgetreten ist (CooErr = -1008).

  - RC

    Returncode des **Device Driver**.

  - Qual

    Wert des Member nqual der vom **Device Driver** zurückgegebenen RESULT-Struktur.

  - Grade

    Wert des Member ngrade der vom **Device Driver** zurückgegebenen RESULT-Struktur.

  - Code

    Wert des Member ncode der vom **Device Driver** zurückgegebenen RESULT-Struktur.

  - Short Descr.

    Anhand des Inhalts der vom **Device Driver** zurückgegebenen RESULT-Struktur aus der **DIT** ausgelesener Beschreibungstext.

5.9 Log-Ansicht

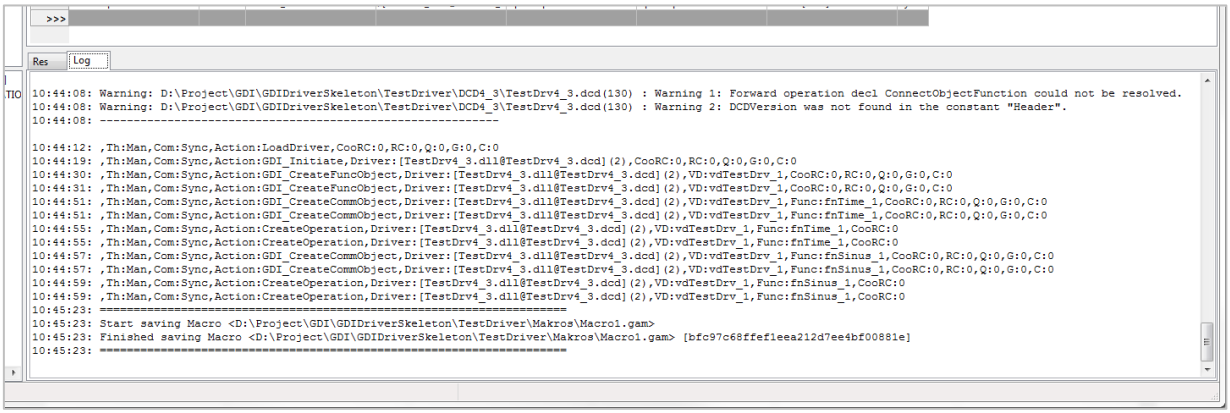


Bild 55: Log-Ansicht

Im rechten unteren Quadranten unter der Karteikarte Log befindet sich die Log-Ansicht mit einem mehrzeiligen Textfeld. Darin werden Log-Informationen angezeigt. Unter anderem werden Informationen über das Parsen der **DCD** sowie eventuelle Fehler dieser und anderer Aktionen ausgegeben.

Die Informationen werden nicht in eine Log-Datei geschrieben, können aber mit den üblichen Tastenkürzeln in diesem Textfeld bearbeitet, gelöscht oder in die Zwischenablage kopiert werden.

5.10 Eingabe von Datenelementen

An dieser Stelle soll noch die Eingabe von Datenelementen beschrieben werden. Die Eingabe wird für **Initiate Parameters** und **Create Parameters**, zum Schreiben von **Parameters** und **Attributes** und für Eingabewerte von **Operations** benötigt.

In den Dialogen zur Eingabe von Datenelementen stehen ständig teilweise die Allgemeinen Funktionen entsprechend Kapitel 5.2 zur Verfügung.

Die Eingabedialoge für **Initiate Parameters** und **Create Parameters** unterscheiden sich geringfügig durch die Möglichkeit Instanz-Namen anzugeben. Dies soll hier nicht weiter beachtet werden. Die Eingabe von Datenelementen ist identisch.

Die Eingabe erfolgt mit Hilfe der bereits vom **DCD**-Baum [5.5] her bekannten Darstellung. Im Gegensatz zum **DCD**-Baum [5.5] beginnt diese Darstellung jedoch bereits mit dem obersten Datenelement des zu bearbeitenden Datenobjekts.

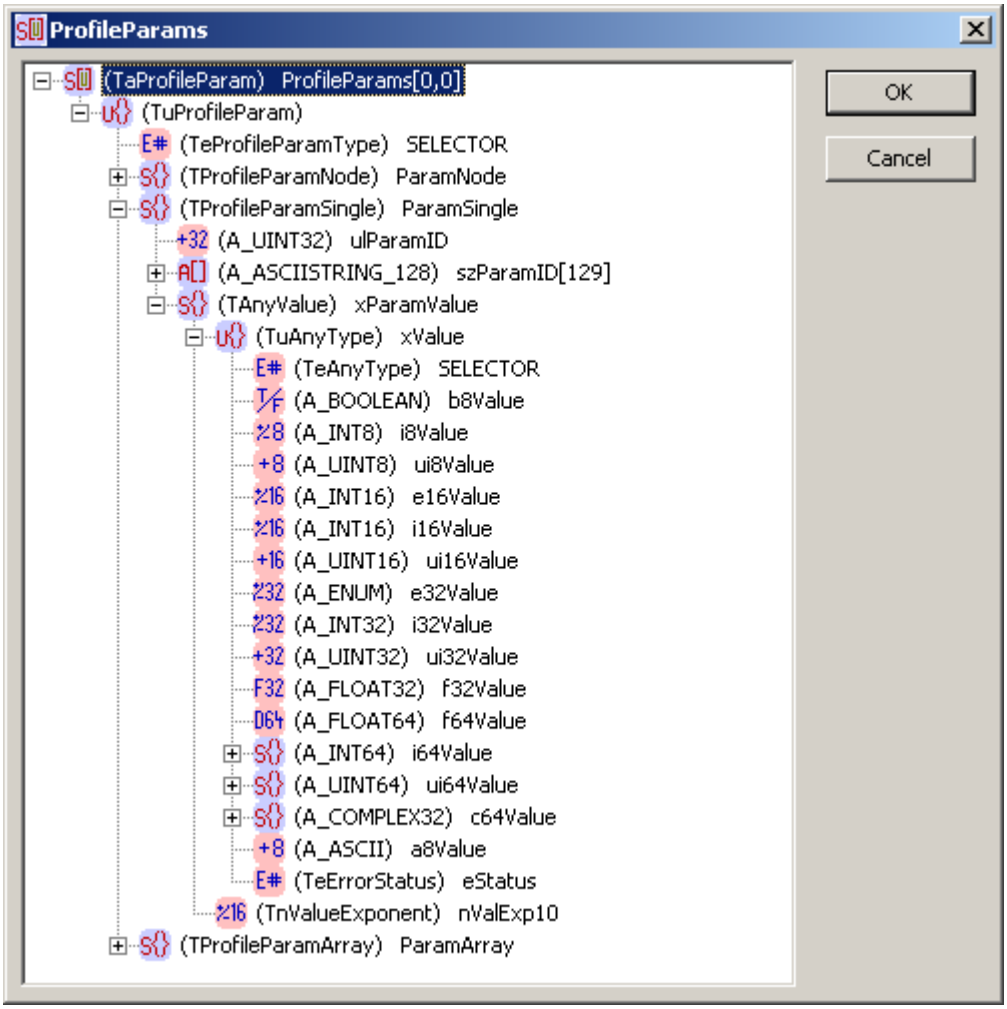


Bild 56: Eingabe von Datenelementen

Das Beispiel zeigt den Eingabewert der **Operation** „SetupProfileParams“ des MDAQ1.2 **Companion**. Diese Datenstruktur enthält fast alle möglichen Datentypen und wird im Folgenden als Beispiel dienen.

Für jeden Datentyp, außer der **Struktur** und der **Union**, kann ein Eingabedialog geöffnet werden. Dies kann entweder durch Doppelklick oder durch betätigen der Leertaste bei markiertem Datenelement erfolgen. Es gibt Dialoge für verschiedene Typen, welche in den nachfolgenden Kapiteln beschrieben werden.

Alle Dialoge sind dabei in zwei Versionen verfügbar. Einer normalen Version die beim Aufruf erscheint und alle regulären Eingaben gestattet und einer Expertenmodus-Version die durch ein Tastenkürzel aufgerufen werden kann. In dieser Version sind erweiterte Eingaben möglich, Sicherungen abgeschaltet oder zusätzliche Informationen verfügbar. Der Expertenmodus wird mit *Alt-E* aktiviert bzw. deaktiviert. Er ist durch einen bläulichen Dialoghintergrund erkennbar.

Abhängig vom jeweiligen Dialog sind weitere Tastenkürzel verfügbar. Welche das sind, sowie Hinweise zur Eingabe, erhalten sie im Menü Help→Hotkeys.

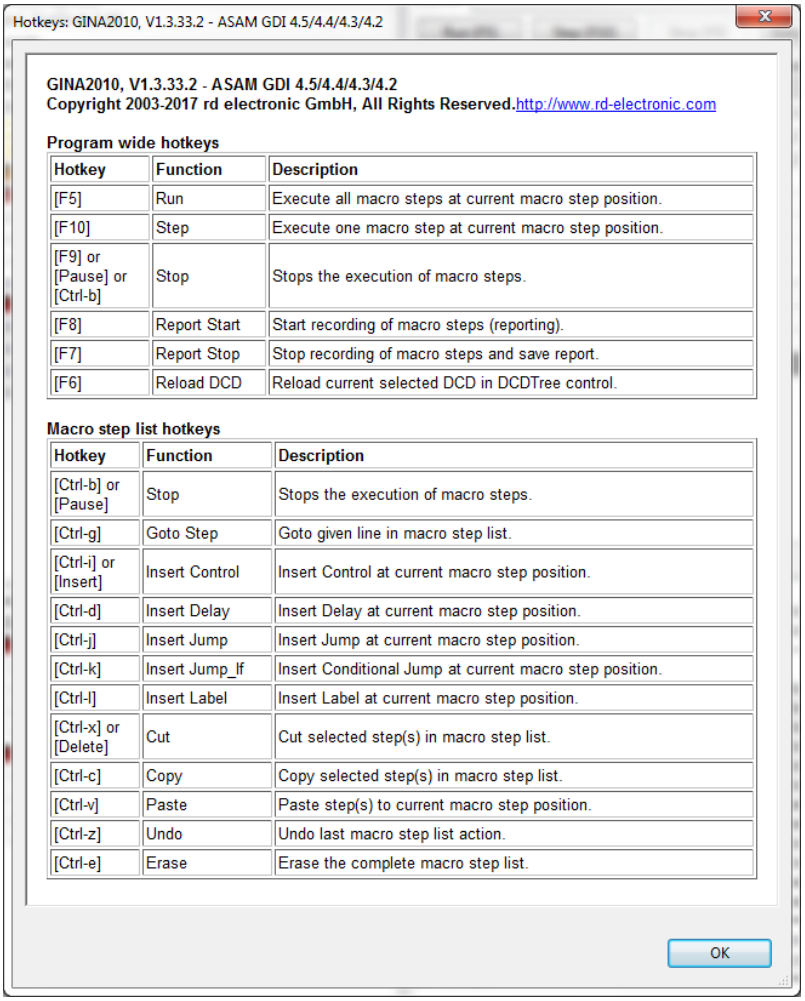


Bild 57: Hotkeys

Werden bei der Eingabe physikalische oder logisch definierte Bereichsgrenzen überschritten, so wird die Schriftfarbe oder die Hintergrundfarbe des Eingabefeldes rot dargestellt. Eine Bestätigung des Dialogs mit OK ist in diesem Zustand nicht möglich.

5.10.1 Sequence

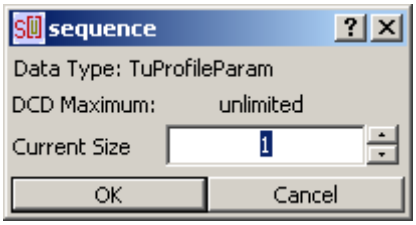


Bild 58: Eingabe Sequence

Mit diesem Dialog kann die aktuelle Länge der **Sequence** erhöht oder verringert werden.

Handelt es sich um eine **Limited Sequence** (nicht dargestellt), so kann die Länge nur bis zum in der **DCD** festgelgten Maximum erhöht werden.

Einmal in den Unterdatentypen eingetragene Werte gehen bei Verlängerung der **Sequence** nicht verloren. Wird die Länge jedoch reduziert, so wird der Inhalt der abgeschnittenen Datenelemente weggeworfen.

Es ist zu beachten, dass bei einer Länge der **Sequence** von `0` kein Speicher reserviert wird. Es wird jedoch trotzdem ein Unterdatentyp angezeigt. Dies dient dazu sich die Datenstruktur auch ohne Dateneingabe betrachten zu können. Die Eingabe von Daten in diesem Zustand ist jedoch nicht möglich. Dazu muss die Länge der **Sequence** erst auf mindestens `1` gesetzt werden.

Fernerhin ist zu beachten, dass bei 8-Bit-Datentypen (**Octet** und **Char**) ein zusätzliches Eingabefeld für Strings erscheint. Dieses ist jedoch noch nicht implementiert und hat keine Wirkung.

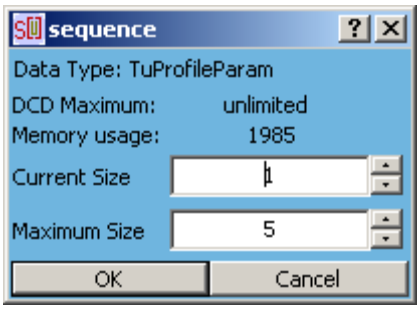


Bild 59: Eingabe Sequence im Expert Mode

Die Darstellung im Expertenmodus unterscheidet sich durch die Möglichkeit, die aktuelle Länge (Current Size) und die maximale Länge (Maximum Size) getrennt setzen zu können.

Zusätzlich wird die durch die maximale Länge verursachte Größe des benötigten Speichers angezeigt. Speicher, den eventuell enthaltene weitere **Sequences** benötigen, wird bei diesem Wert jedoch nicht berücksichtigt.

5.10.2 Array

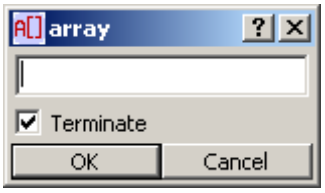


Bild 60: Eingabe Octet/Char Array

Dieser Dialog erscheint nur bei 8-Bit-Datentypen (**Octet** und **Char**) als Inhalt des **Array**. Für alle anderen Datentypen ist kein spezieller Eingabedialog für **Arrays** verfügbar. Die Eingabe der Werte hat in den unterhalb des **Array** angezeigten Datenelementen zu erfolgen.

In dem Eingabefeld kann ein String in das **Array** eingegeben werden. Der String wird nach Bestätigung des Dialogs mit OK in das **Array** kopiert. Der möglicherweise verbleibende Rest des **Array** wird mit 0-Bytes aufgefüllt.

Durch die Eingabe des Zeichens ‚^' gefolgt von zwei hexadezimalen Ziffern können beliebige, nicht auf der Tastatur verfügbare, Zeichen eingegeben werden.

Ist „Terminate" gesetzt, so wird sichergestellt, dass ein 0-Byte als Terminierung angehängt wird, wenn das letzte Zeichen nicht bereits ein 0-Byte ist.

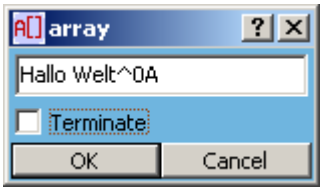


Bild 61: Eingabe Octet/Char Array mit 0-Terminierung

Der Expertenmodus gestattet im Falle des **Array** Dialogs keine anderen Eingaben als im normalen Modus.

5.10.3 Integer

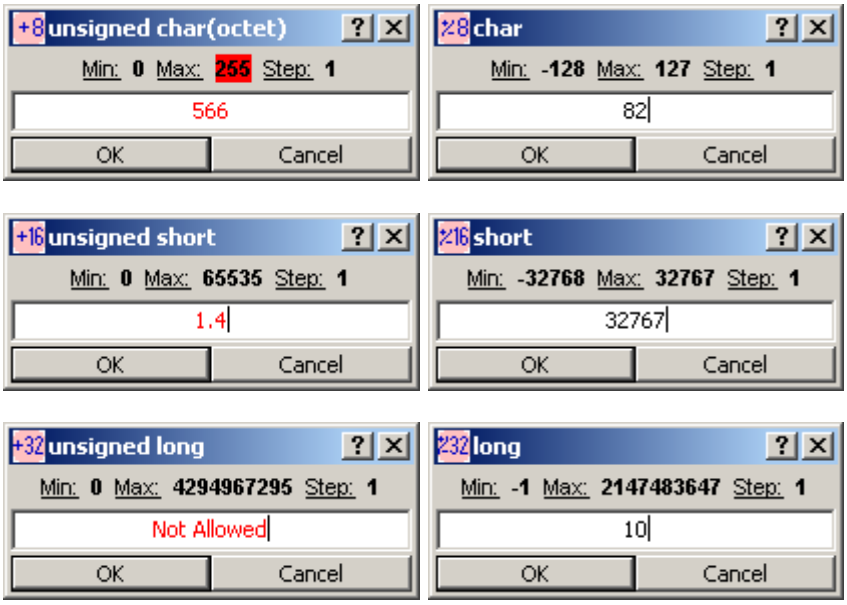


Bild 62: Eingabe Integer-Werte

Die oben aufgeführten Dialoge dienen zur Eingabe von Integer-Werten. Die dargestellten Beispiele zeigen zur Veranschaulichung sowohl korrekte als auch fehlerhafte Eingaben.

Oberhalb des Eingabefeldes werden die Gültigkeitsbegrenzungen (Validity) angezeigt. Kann bei einem fehlerhaften Eingabewert festgestellt werden welcher Gültigkeitswert verletzt wurde, so wird dieser rot markiert.

Die Eingabe der Zahlenwerte kann auf verschiedene Art und Wiese erfolgen. Die folgende Tabelle zeigt die Eingabemöglichkeiten beispielhaft für diesen Dialog. Diese Information ist für jeden Dialog wie, oben beschrieben, als Hilfe verfügbar.

Format	**Tastenkürzel zur Umwandlung (z.B. Alt-D)**	**Formatbeschreibung**
Dezimal	D | d	*[D] | *[d]
Hexadezimal	X | H | x | h	[0]X* | *H | [0]x* | *h
Oktal	O | o | 0	0*
Binär	B | b	*B | *b

Zeichen (Nur **Char** und **Octet**)		´?´

Tabelle 7: Integer Anzeige-Formate

Der Expertenmodus unterscheidet sich bezüglich der verfügbaren Bedienelemente nicht vom normalen Modus. Die Gültigkeitsgrenzen aus der **DCD** werden jedoch nicht mehr überprüft. Lediglich die physikalischen Grenzen des entsprechenden Datentyps begrenzen die Eingabe.

5.10.4 Real

Bild 63: Eingabe Real-Werte

Die oben aufgeführten Dialoge dienen zur Eingabe von **Float**- und **Double**-Werten.

Oberhalb des Eingabefeldes werden die Gültigkeitsbegrenzungen (Validity) angezeigt. Kann bei einem fehlerhaften Eingabewert festgestellt werden welcher Gültigkeitswert verletzt wurde, so wird dieser rot markiert.

Die Eingabe der Zahlenwerte kann auf verschiedene Art und Wiese erfolgen. Die verschiedenen Eingabemöglichkeiten für diesen Dialog sind in der oben beschrieben Form als Hilfe verfügbar.

Es ist zu beachten, dass die Eingabe in hexadezimaler oder binärer Form nicht den inhaltlichen Wert des entsprechenden Realwertes setzt, sondern direkt die Speicherzelle beschreibt. Für eine sinnvolle Eingabe ist deshalb Kenntnis über das binäre Format des entsprechenden Realwertes erforderlich.

Der Expertenmodus unterscheidet sich bezüglich der verfügbaren Bedienelemente nicht vom normalen Modus. Die Gültigkeitsgrenzen aus der **DCD** werden jedoch nicht mehr überprüft. Lediglich die physikalischen Grenzen des entsprechenden Datentyps begrenzen die Eingabe.

5.10.5 Enumeration

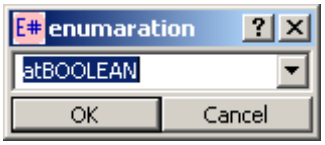


Bild 64: Eingabe Enumerator

Dieser Dialog dient zur Eingabe von **Enumerations**.

Die verfügbaren Member der **Enumeration** (**Enumeratoren**) können in der Drop Down Box ausgewählt werden. Die Darstellung der dahinterliegenden numerischen Werte ist mittels Tastenkürzel *Alt-D* möglich.

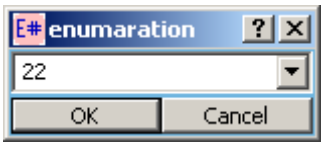


Bild 65: Eingabe Enumerator im Expert Mode

Der Expertenmodus unterscheidet sich bezüglich der verfügbaren Bedienelemente nicht vom normalen Modus.

Es ist jedoch möglich dezimale Werte einzugeben. Dadurch können auch **Enumeratoren** gesetzt werden, die nicht in der Liste aufgeführt sind. Die möglichen Grenzen entsprechen einem **Short**-Wert der bei **GDI** als zugrundeliegender Datentyp für **Enumerations** dient.

5.10.6 Boolean

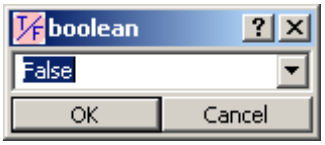


Bild 66: Eingabe Boolean

Dieser Dialog dient zur Eingabe von booleschen Werten.

In **GDI** ist ein **Boolean** als `unsigned char` (**GDI** Datentyp **Octet**) dargestellt. Neben den üblichen Werten von `0` für `False` und `1` für `True` können auch Werte von `2` bis `255` vorkommen.

Die Eingabe kann entweder durch Auswahl in der Drop Down Box oder durch eines der folgenden Tastenkürzel erfolgen.

Boolscher Wert	Numerischer Wert	Tastenkürzel (z.B. Alt-T)
`True`	`1`	T, t, W, w, Y, y, J, j
`False`	`0`	F, f, N, n

Tabelle 8: Tastenkürzel für Eingabe Boolean

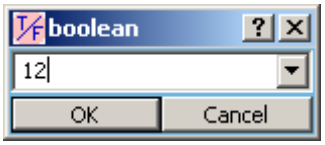


Bild 67: Eingabe Boolean im Expert Mode

Auch hier unterscheidet sich der Expertenmodus bezüglich der verfügbaren Bedienelemente nicht vom normalen Modus.

Es ist jedoch möglich dezimale Werte einzugeben. Dazu ist das Tastenkürzel *Alt-D* zu benutzen. Der mögliche Wertebereich entspricht, wie bereits oben erwähnt, einem `unsigned char` (**GDI** Datentyp **Octet**). Es ist zu beachten, dass sobald mit *Alt-S*, oder auf andere Weise, zur Stringdarstellung zurückgeschaltet wird, der interne numerische Wert wieder auf `1` gesetzt wird.

Der Wert `1` für `True` wurde gewählt, weil die Praxis gezeigt hat, dass viele **Device Driver** nur auf `1` für `True` reagieren. **GDI** erlaubt jedoch alle Werte ungleich `0` zur Darstellung von `True` [GDI-Doc3].

Es ist außerdem möglich in diesem Modus den gewünschten Wert auch als String einzugeben. Die folgende Tabelle zeigt die möglichen Strings. Die Auswertung erfolgt dabei ohne Beachtung von Groß- und Kleinschreibung.

Boolescher Wert	Numerischer Wert	Eingabestrings
`True`	`1`	„true“ „wahr“ „yes“ „ja“ „ok“
`False`	`0`	„false“ „falsch“ „no“ „nein“ „cancel“

Tabelle 9: Eingabe Boolean mittels Strings

5.10.7 Union

Der Datentyp **Union** besitzt zwar keinen eigenen Dialog, soll aber trotzdem hier kurz erwähnt werden.

Eine **Union** besteht aus dem **Selector** und den einzelnen Membern. Der **Selector** selbst kann dabei ein beliebiger Integer-Typ, eine **Enumeration** oder gar ein Boolescher Wert sein. Jeder Member ist einem bestimmten Wert des **Selector** zugeordnet. Es sollten nur Daten in den durch den Wert des **Selector** aktivierten Zweig der **Union** eingegeben werden.

Eingabe in einen falschen Zweig der **Union** führt zwar nicht zu einem Absturz, aber zu unerwarteten Werten im durch den **Selector** ausgewählten Zweig. Es soll hier noch erwähnt werden, dass **GDI** die Verwendung von **Sequencen** innerhalb von **Unions** verbietet. Dadurch ist es nicht möglich, durch die Benutzung falscher Zweige, auf undefinierte **Sequence** Pointer zuzugreifen.

5.11 Information Report

Der **Information Report** ist eine **GDI** Funktionalität, die es gestattet, den Datentransfer von Seite des **Device Driver** aus einzuleiten. Dabei wird ein meist zyklischer Mechanismus im **Device Driver** aktiviert, der eine CallBack-Funktion in der **Coordinator Engine** aufruft. Die **Coordinator Engine** leitet diesen Aufruf dann auf geeignete Weise an die Anwendung weiter. Ein derartiger Aufruf ist immer an eine Instanz eines bestimmten **Communication Object** gebunden. Dadurch ist es möglich verschiedene **Information Report** Mechanismen zu unterscheiden.

GINA2010 erzeugt für jeden **Information Report** einer bestimmten Instanz eines **Communication Object** ein eigenes Dialogfenster. Dieses wird erzeugt, sobald der Erste **Information Report** auftritt. Alle weiteren **Information Reports** derselben Instanz aktualisieren nun diesen Dialog. Wird der Dialog geschlossen, so öffnet der nächste **Information Report** ihn erneut.

Das folgende Bild zeigt ein Beispiel.

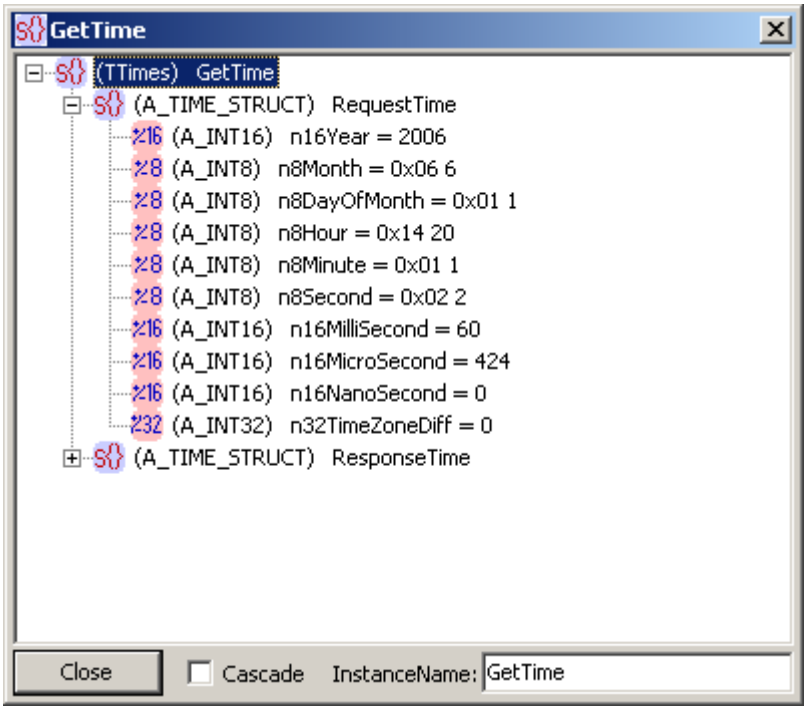


Bild 68: Information Report

Im Textfeld `InstanceName` wird der Name der zugehörigen Instanz eines **Communication Object** angezeigt.

Der Expansionszustand des Baums bleibt auch während der Aktualisierung erhalten.

Kann die Aktualisierung des Baums nicht innerhalb eines Intervalls des **Information Report** abgeschlossen werden, so wird der nächste **Information Report** nicht blockiert sondern sofort zur **Coordinator Engine** zurückgegeben. Dabei wird jedoch der letzte Dateninhalt zwischengespeichert, um die Daten des letzten **Information Report** darstellen zu können, sobald ausreichend Zeit dafür ist. Der Dialog sollte somit immer die aktuellsten Daten enthalten.

Ist `Cascade` gesetzt, so werden alle eintreffenden **Information Reports** einer Instanz eines **Communication Object** im gleichen Fenster untereinander dargestellt. Dies ist nur sinnvoll, wenn die Anzahl der **Information Reports** innerhalb eines bestimmten Zeitraumes begrenzt bzw. überschaubar ist.

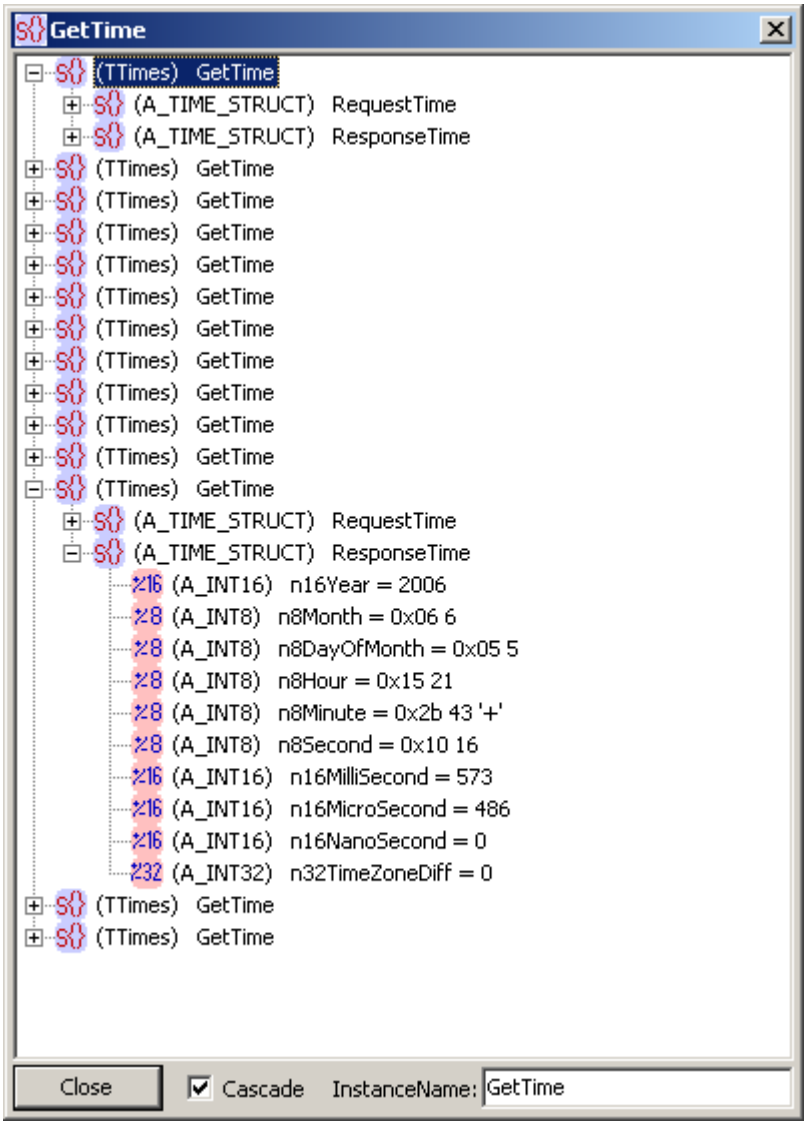


Bild 69: Information Report "Cascade"

Jeder auftretende **Information Report** wird ebenfalls im Log-Ansicht [5.9] gelistet. Auf Grund von Einschränkungen in der Benutzung der grafischen Oberfläche von Windows werden sowohl die Daten für den **Information Report** Dialog, als auch für die Log-Ansicht, über eine Message-Queue geleitet. Im Gegensatz zur Aktualisierung des Dialogs wird jedoch jeder **Information Report** in der Log-Ansicht gelistet. Das kann dazu führen, dass das Programm nach Beenden des **Information Report** noch einige Sekunden Volllast erzeugt. In dieser Zeit werden eventuell aufgestaute Log-Einträge verarbeitet.

6 Skeleton Generator – Programmbeschreibung

Der Skeleton Generator lässt sich entsprechend Kapitel 5.5.5 über das Kontextmenü eines **DCD**-Kontens im **DCD**-Baum [5.5] öffnen.

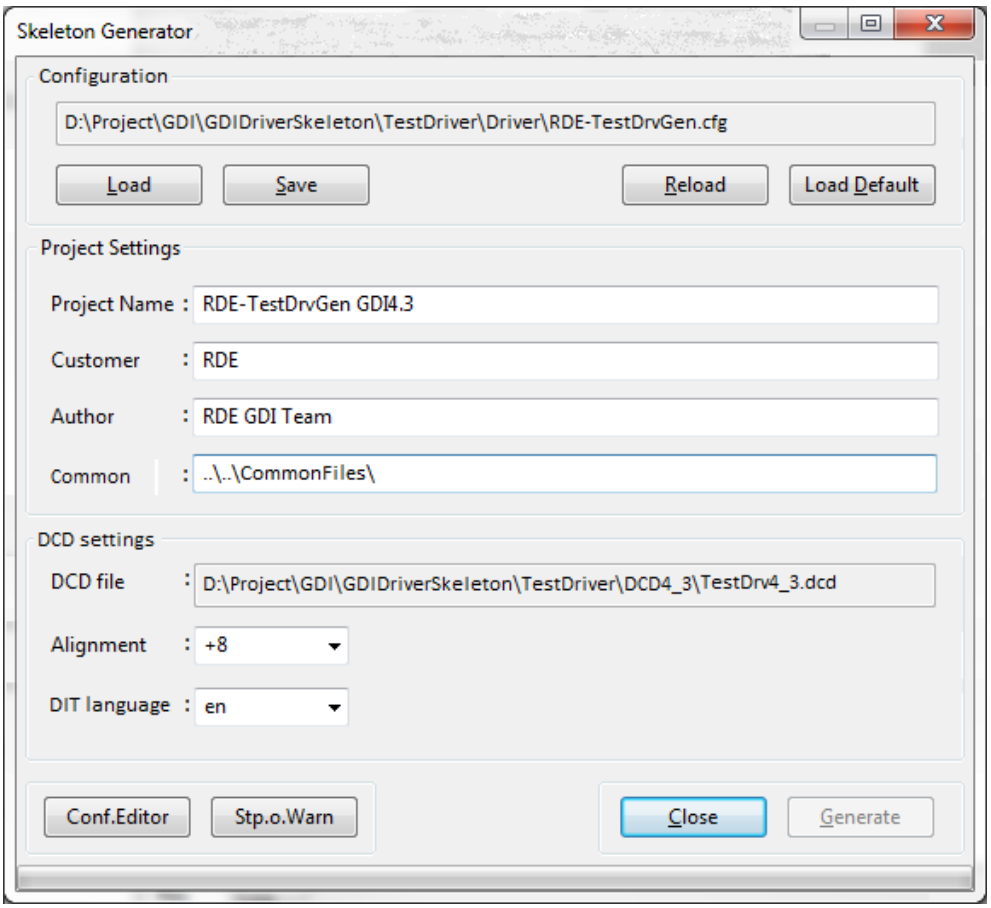


Bild 70: Skeleton Generator – Hauptdialog

RDE bietet als integriertes Entwicklungswerkzeug den **Skeleton Generator** zur Generierung von **Device Driver Skeletons** an. Dabei wird entsprechend der ausgewählten **DCD** ein **Device Driver Skeleton** generiert, welcher durch **Anlagenlieferanten** zu einem vollwertigen **Device Driver** ausgebaut werden kann.

Device Driver Skeletons implementieren dabei alle Konstrukte entsprechend der geladenen **DCD** nach dem **ASAM GDI** Standard, sodass sich **Anlagenlieferanten** nur in einen recht kleinen Teil des komplexen **ASAM GDI** Standards einarbeiten müssen, und sich so bei der Entwicklung eines **Device Driver** für eine Anlage oder ein Gerät in wirtschaftlicher Form auf ihre Kernaufgabe konzentrieren können.

Der **Skeleton Generator** unterstützt weiterhin die wiederholte Generierung eines **Device Driver Skeleton**, welcher mit dem hier vorliegenden **Skeleton Generator** bereits geniert und durch den **Anlagenlieferanten** befüllt wurde. Um dies zu ermöglichen, enthalten die generierten Quelldateien gekennzeichnete Sektionen, welche der **Skeleton Generator** bei einer Neugenerierung verarbeitet und so in den neu generierten **Device Driver** einfließen lässt.

Für die Benutzung des **Skeleton Generator** ist der Kauf einer entsprechenden Lizenz [2.2] erforderlich.

Der nach Öffnen des **Skeleton Generator** erscheinende Dialog [Bild 70] bietet die in den folgenden Kapiteln beschriebenen Funktionalitäten.

6.1 Konfiguration

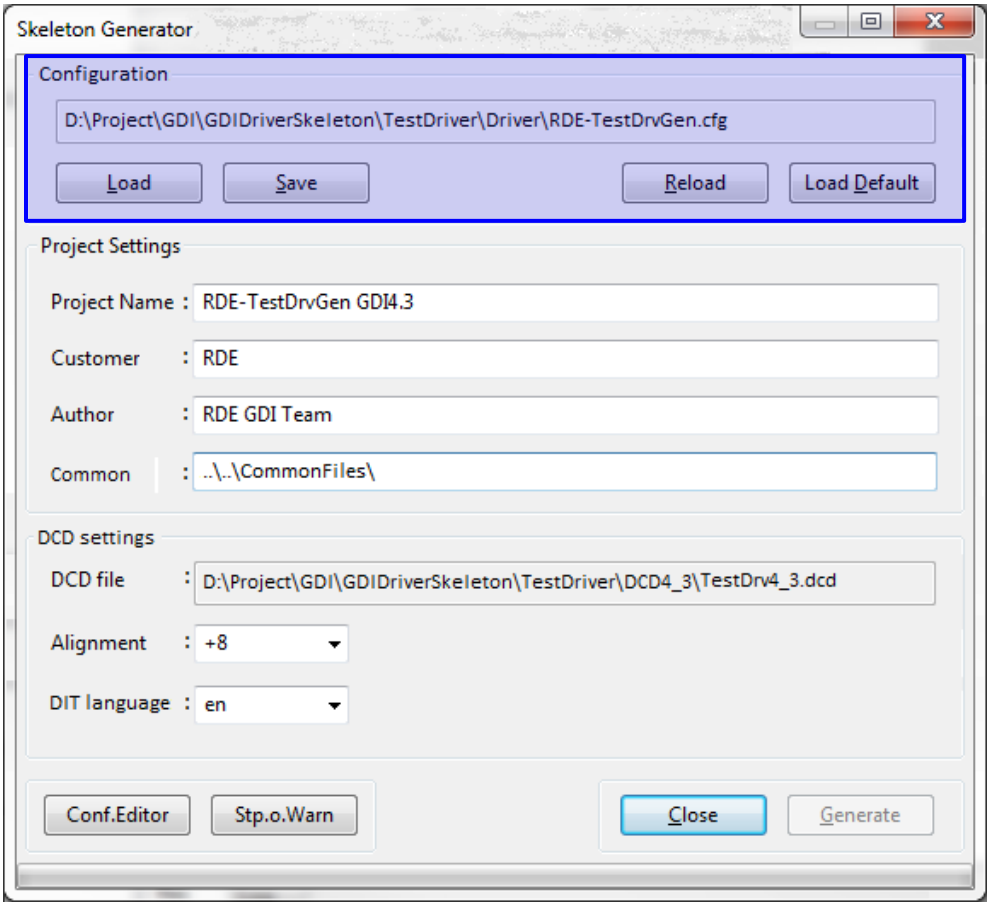


Bild 71: Skeleton Generator – Konfigurationsdatei

Der **Skeleton Generator** unterstützt das Speichern und Laden von Konfigurationsdateien, welche die im mittleren Bereich vorgenommenen Projekteinstellungen [6.2] und DCD-Einstellungen [6.3] zur Generierung von **Device Driver Skeletons** enthalten. Dies erspart dem Benutzer die Neu-Eingabe der Projekteinstellungen bei wiederholter Generierung bereits generierter und entwickelter **Device Drivers**.

Die aktuell geladene Konfigurationsdatei wird im Textfeld der Gruppe `Configuration` angezeigt. Der Pfad wird entsprechend nach dem Laden (`Load`) und Speichern (`Save`) von Konfigurationen angepasst und dargestellt.

- Load

  Durch Anwahl des Schalters Load öffnet sich ein Systemdialog zur Auswahl einer vorhandenen Konfigurationsdatei.

  Durch das Laden einer Konfigurationsdatei werden die Projekteinstellungen aus dieser Datei in den Dialog übernommen [6.2].

  Das Textfeld der Gruppe Configuration wird nach dem Laden entsprechend aktualisiert.

- Save

  Durch Anwahl des Schalters Save öffnet sich ein Systemdialog zur Eingabe eines Dateipfades für die zu speichernde Konfigurationsdatei.

  Durch das Speichern einer Konfigurationsdatei werden die Projekteinstellungen [6.2] in dieser Datei gespeichert.

  Das Textfeld der Gruppe Configuration wird nach dem Speichern entsprechend aktualisiert.

- Reload

  Durch Anwahl des Schalters Reload wird die bereits geladene und im Textfeld der Gruppe Configuration dargestellte Konfigurationsdatei neu eingelesen und die Projekteinstellungen aus dieser Datei in den Dialog übernommen [6.2].

- Load Default

  Durch Anwahl des Schalters Load Default werden die Standard-Projekteinstellungen geladen und in den Dialog übernommen [6.2].

6.2 Projekteinstellungen

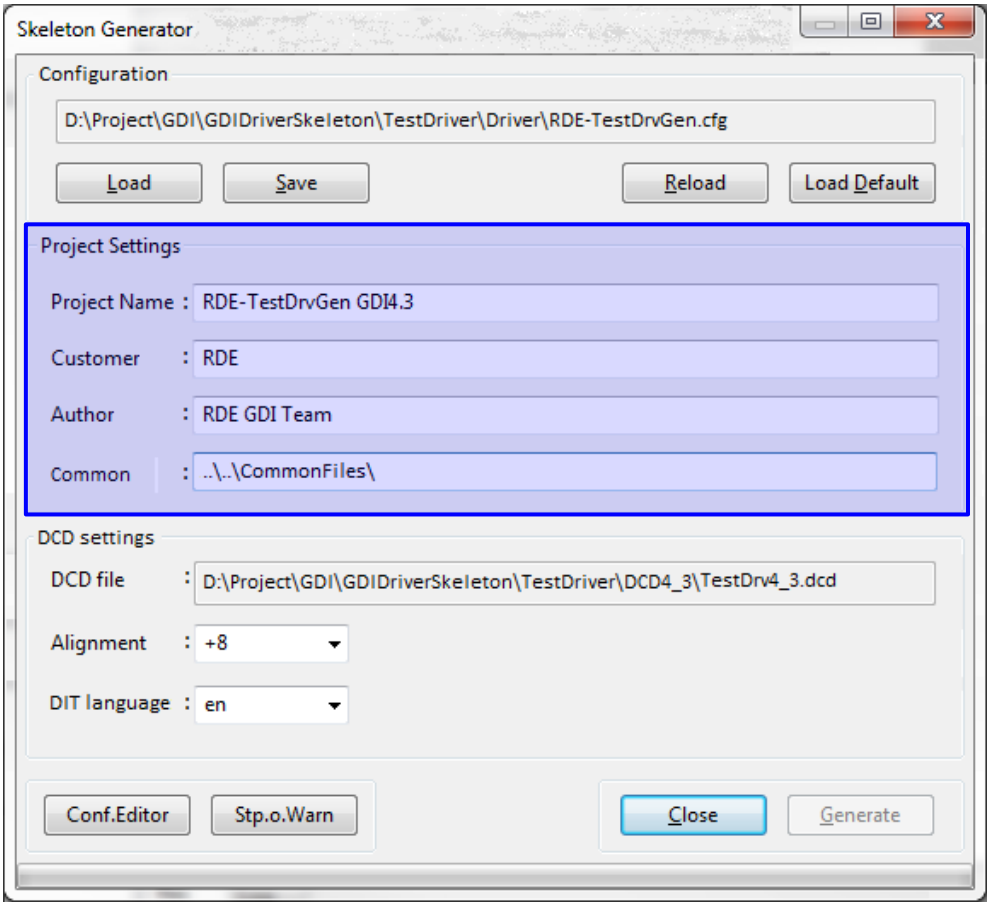


Bild 72: Skeleton Generator – Projekteinstellungen

Die Informationen `Project Name`, `Customer` und `Autor` werden lediglich als im C++-Format kommentierte Kopfinformationen in die zu generierenden Quelldateien geschrieben.

Der unter Common anzugebende Pfad muss auf das Verzeichnis zeigen, in welchem sowohl allgemeine Quellen der Firma **RDE** als auch Template-Dateien für den Generierungsvorgang zu finden sind.

Das Verzeichnis wird nach Erwerb einer Lizenz [2.2] des Skeleton Generator durch **RDE** bereitgestellt.

> ⚠ Die mitgelieferten allgemeinen Quellen der Firma **RDE** dürfen ausschließlich durch **GINA2010** zur Generierung von **Device Driver Skeletons** und deren Erstellung zu **Device Drivers** aus den mittels **GINA2010** genierten Entwicklungsprojekten genutzt werden.
>
> Eine anderweitige Benutzung dieser Quellen ist untersagt.

6.3 DCD-Einstellungen

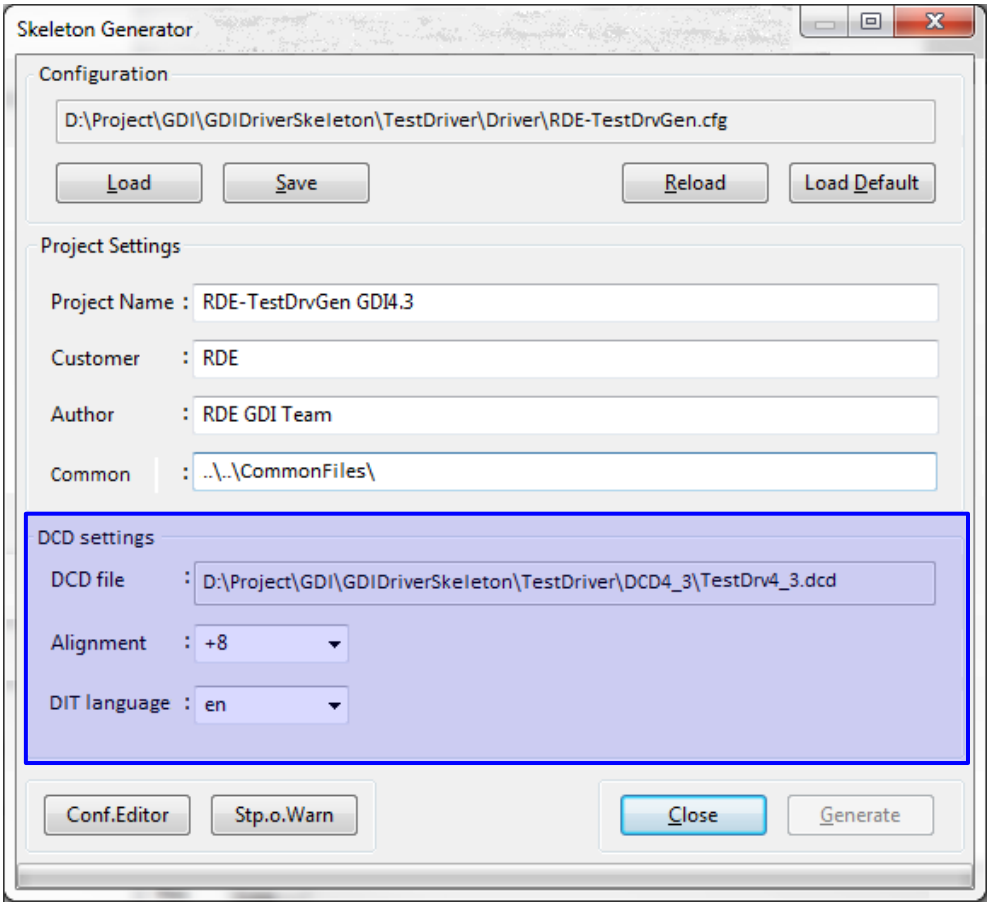


Bild 73: Skeleton Generator – Einstellungen zur DCD

- DCD file

  Der unter DCD file aufgeführte Inhalt zeigt den vollständigen Dateipfad, der **DCD**-Datei deren Knoten im **DCD**-Baum [5.5] ausgewählt und über dessen Kontextmenü der **Skeleton Generator** geöffnet wurde.

  Ist die ausgewählte **DCD** nicht die zur Generierung gewünschte **DCD**, ist der **Skeleton Generator** zu schließen [6.5], die richtige **DCD** in **GINA2010** zu laden [5.5], und der **Skeleton Generator** neu zu starten [5.5.5].

- Alginment

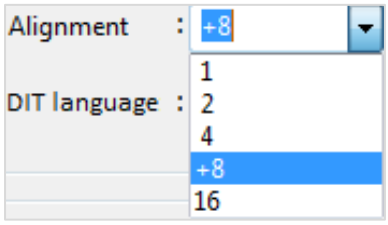


Bild 74: Skeleton Generator: Einstellen des DCD-Alignment

Unter Alignment erlaubt eine Combobox die Auswahl des Alignments, mit welchem der zu generierende **Device Driver** seine **DCD**-Strukturen [4.1.3] übersetzen wird.

- DIT language

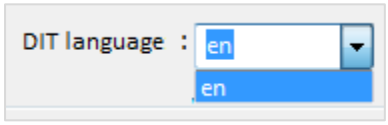


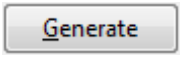
Bild 75: Skeleton Generator: Auswahl der DIT-Sprache

Unter DIT language erlaubt eine Combobox die Auswahl einer von der ausgewählten **DCD** unterstützten **DIT**-Sprache.

Der **Skeleton** Generator verwendet im generierten **Device Driver Skeleton** Teile der **DIT**-Texte für eine gute Erkennung von Fehlercode-Variablen, welche aus den **DIT**-Inhalten generiert werden.

Ansonsten hat die ausgewählte **DIT**-Sprache keinen Einfluss auf den **Device Driver**. Vielmehr findet die Verwendung von **DIT**-Sprachen erst zur Laufzeit des **Device Driver** durch die **Coordinator Engine** statt, welche der Anwendung Meldungs-, Hinweis- und Fehlertexte aus dem **Device Driver** in der zur Laufzeit einzustellenden **DIT**-Sprache [5.5.2.4] präsentiert.

6.4 Genierung eines Device Driver Skeleton

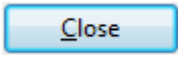


Durch Anwahl des Schalters Generate geniert der **Skeleton Generator** ein **Device Driver Skeleton** entsprechend den Projekteinstellungen [6.2] und den DCD-Einstellungen [6.3]. Dieses besteht aus einem C++-Projekt für Microsoft Visual Studio 6.0, einem Makefile für Unix-Systeme und C++- Quelldateien entsprechend der ausgewählten **DCD**.

Die generierten C++-Quelldateien sind dabei mit gekennzeichneten Sektionen versehen, welche nun durch den Entwickler des **Device Driver** befüllt werden können.

Eine Dokumentation zur Benutzung der generierten Inhalte ist im Dokument [RDE-Doc4] zu finden.

6.5 Schließen des Skeleton Generator



Der Dialog des Skeleton Generator kann mittels des Schalters Close bzw. über die übliche Dialogsteuerung X geschlossen werden.

Falls vorher eine Konfigurationsdatei mittels Load geladen [6.1] und daraufhin Projekteinstellungen [6.2] oder DCD-Einstellungen [6.3] geändert wurden, erfolgt eine Sicherheitsabfrage mit dem Hinweis auf eine nicht gespeicherte Konfiguration.

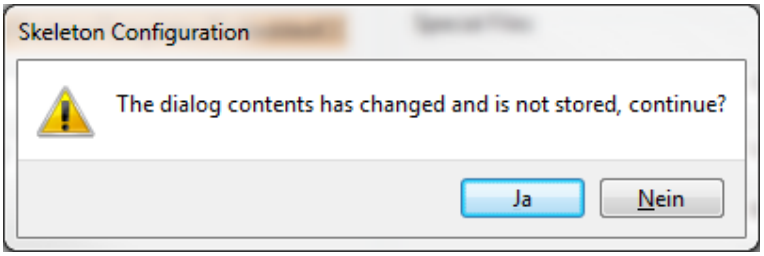


Bild 76: Skeleton Generator: Sicherheitsabfrage für ungesicherte Konfiguration

Bestätigt der Benutzer diesen Dialog mittels Schalter Ja, wird der **Skeleton Generator** ohne eine Sicherung der aktuellen Konfiguration geschlossen.

Bei Anwahl des Schalters Nein, bleibt der **Skeleton Generator** weiterhin geöffnet, sodass eine Sicherung mittels Schalter Save [6.1] vorgenommen werden kann.

7 Ablaufbeispiel mit dem RDE Testdriver

Das folgende Beispiel soll den Ablauf eines Tests mit einem **Device Driver** und **GINA2010** demonstrieren. Es wird anhand des **RDE Testdriver** durchgeführt.

Der **RDE Testdriver** ist ein **Device Driver** nach **GDI** 4.3.2 [GDI-Doc3] der kein Gerät benötigt.

Er implementiert zwei **Interfaces** mit **Communication Objects** zum Lesen von Zeitwerten und einer Sinus Sequenz. Zusätzlich ist es möglich, die Werte über **Information Reports** auszugeben.

Der **RDE Testdriver** benötigt einen **Platform Adapter** nach **GDI** 4.3 jedoch keine **Platform Adapter Extension**.

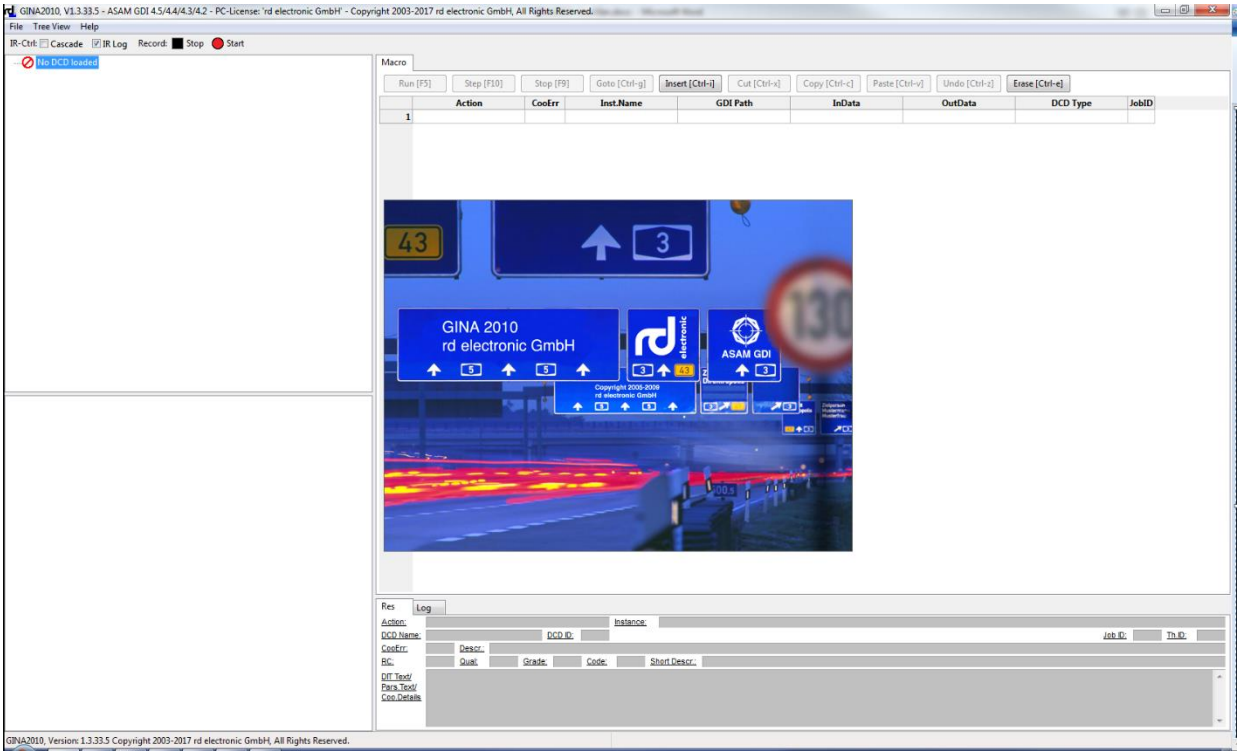
Das Beispiel führt dabei durch die Benutzung des **DCD**-Baums [5.5] und des Instanzen-Baums [5.6].

Während der Durchführung werden ständig weitere Einträge in der Schrittliste [5.7], in der Ergebnisansicht [5.8] und in der Log-Ansicht [5.9] erzeugt, auf welche hiermit generell verwiesen sei.

7.1 Start von GINA2010

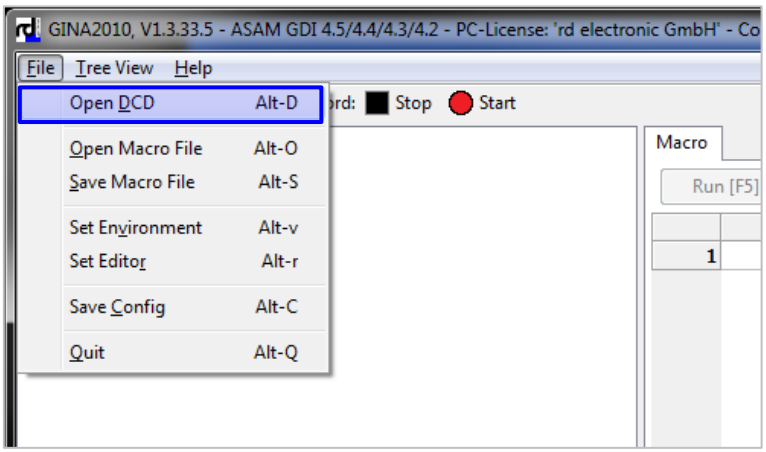
Es wird die Anwendung **GINA2010** in einer durch das zugrundeliegende Betriebssystem unterstützten Weise, z.B. durch Doppelklicken der Anwendung im Dateisystem, gestartet.

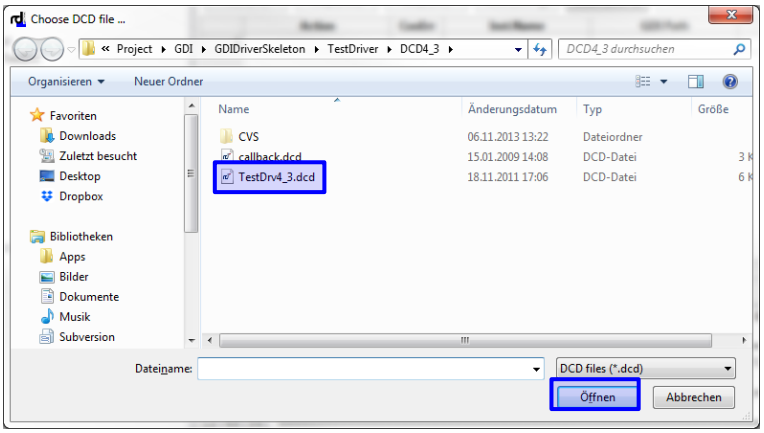
Es öffnet sich das Hauptfenster der Anwendung, welches zur Laufzeit ständig zu sehen ist.



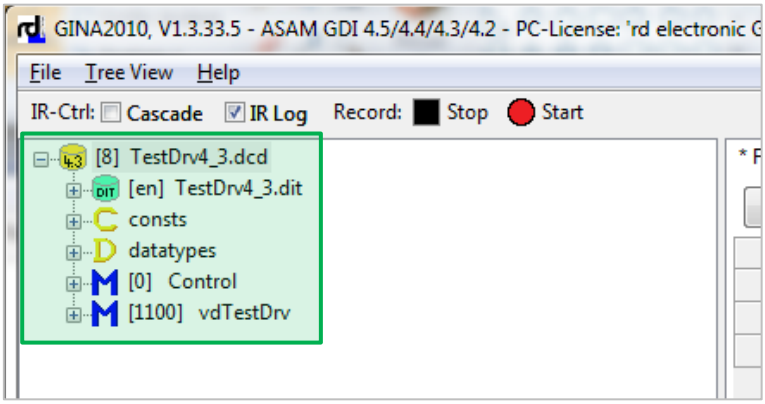
7.2 Laden der DCD

Zuerst wird entsprechend Kapitel [5.3.1] wird die **DCD**-Datei geladen.



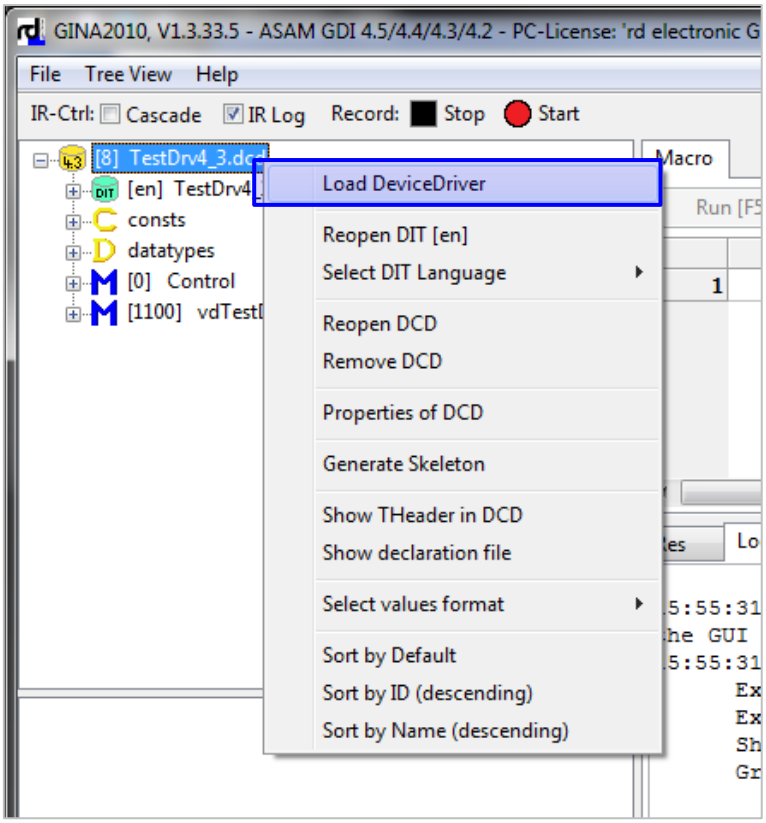


Im Ergebnis erscheint die Root-Instanz im **DCD**-Baum [5.5], welche die DCD präsentiert.



7.3 Laden des Device Driver

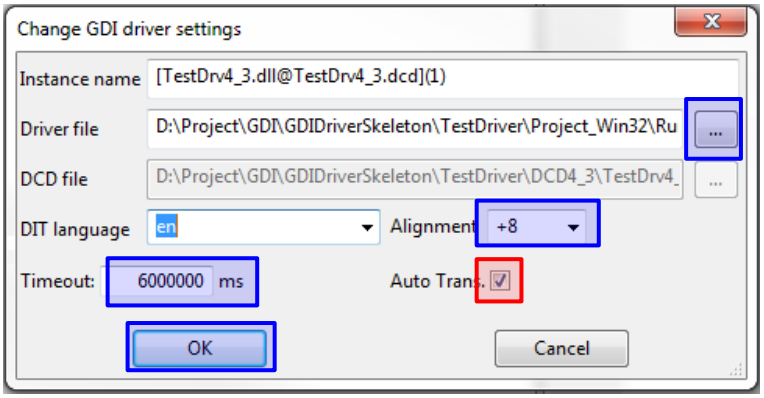
Entsprechend Kapitel [5.5.2] ist nun der **Device Driver** zu parametrieren und zu laden.



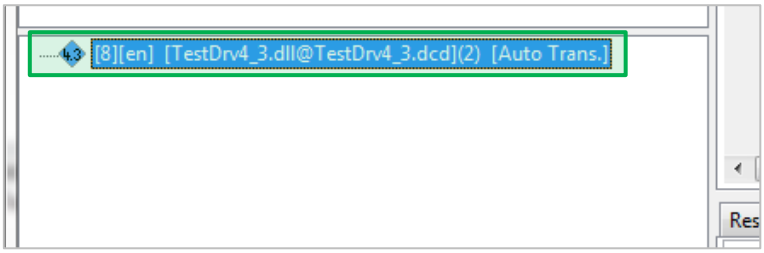
Es erscheint der Dialog zur Parametrierung [5.5.2] des **Device Driver**.

7.3.1 Automatischer Betriebsmodus

Im Beispiel wird der **Device Driver** mittels ... [5.5.2.2] ausgewählt, das **DCD**-Alignment [5.5.2.5] auf +8 eingestellt, das Timeout [5.5.2.6] für die Ausführung von **Operationen** auf 100 Minuten eingestellt, der Automatische **Betriebsmodus** [5.5.2.7] aktiviert und der Dialog mittels OK bestätigt.



Im Ergebnis erscheint die **Device Driver** Instanz im Instanzen-Baum [5.6]. Die Instanz enthält den Zusatz „[Auto Trans.]“ als Hinweis auf den Automatischen **Betriebsmodus**.



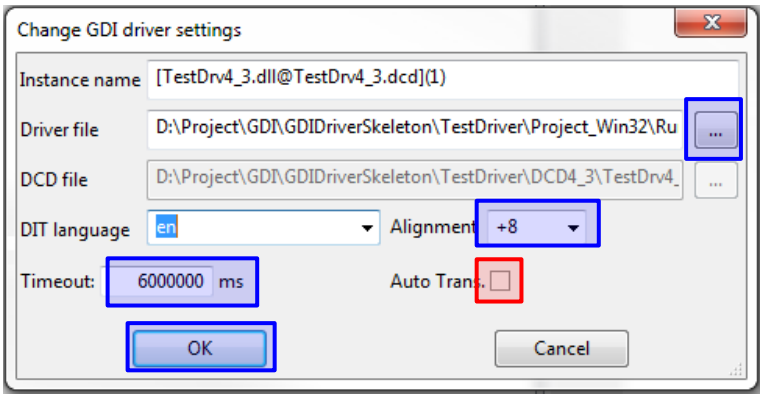
Bei Verwendung des Automatischen **Betriebsmodus** werden die **GDI Phasen**-Übergänge durch die integrierte **Coordinator Engine** bedarfsgerecht automatisch herbeigeführt.

7.3.2 Manueller Betriebsmodus

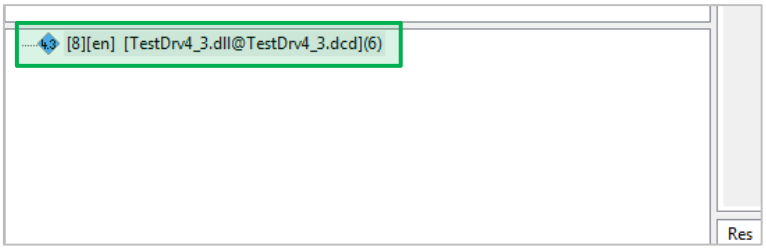
Normalerweise ist es nicht notwendig den Automatischen **Betriebsmodus** zu verlassen. Es gibt jedoch eine Information, die nur im Manuellen **Betriebsmodus** [5.5.2.7] zur Verfügung steht.

Der `GDI_Identify` Aufruf des **Device Driver** muss immer mit einem **Virtual Device** als Parameter ausgeführt werden. Dies ist für manuell angelegte **Virtual Device** Instanzen möglich [7.4.1.1]. **GDI** definiert aber, dass ein `GDI_Identify`, mit dem **Control VD** als Parameter, Informationen zurückliefert, die für den gesamten **Device Driver** gültig sind. Um diese Informationen abfragen zu können, muss das **Control VD** manuell angelegt werden.

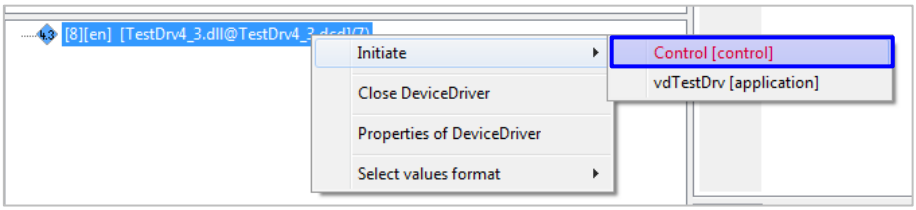
Somit wird der **Device Driver** mittels `...` [5.5.2.2] ausgewählt, das **DCD**-Alignment [5.5.2.5] auf `+8` eingestellt, das Timeout [5.5.2.6] für die Ausführung von **Operationen** auf `100` Minuten eingestellt, der Manuelle **Betriebsmodus** [5.5.2.7] `aktiviert` und der Dialog mittels `OK` bestätigt.



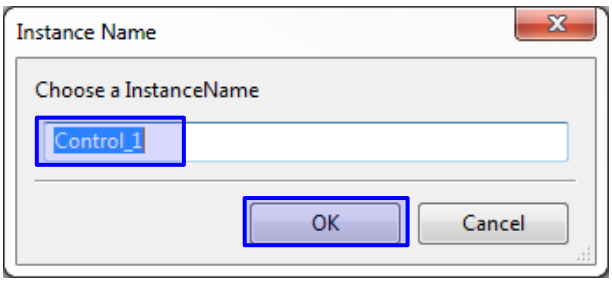
Im Ergebnis erscheint die **Device Driver** Instanz im Instanzen-Baum [5.6].



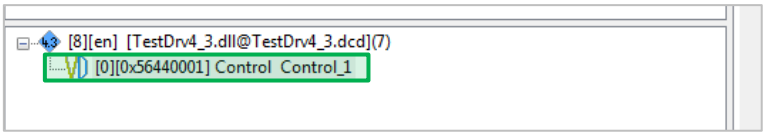
Als ersten Schritt im Manuellen **Betriebsmodus** wird das **Control VD** angelegt.



Es erfolgt die Abfrage des Instanz-Namens für das zu erstellende **Virtual Device**. Ein eindeutiger Vorschlag ist bereits in den erscheinenden Dialog eingetragen, kann aber auch geändert werden. Die vergebenen Namen müssen eindeutig sein.



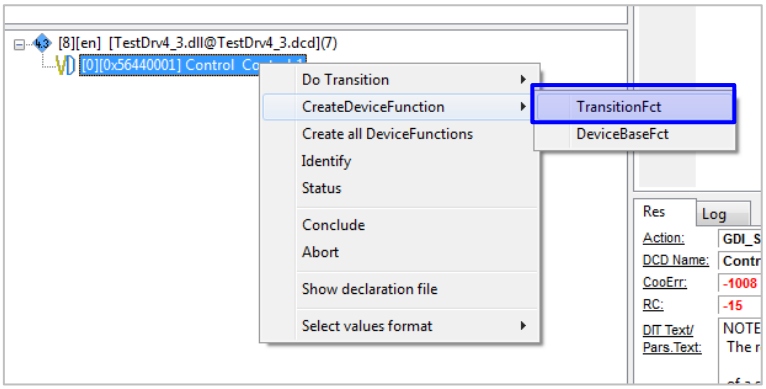
Nach Bestätigung durch Drücken des Schalters OK wird im Instanzen-Baum [5.6] der folgende Eintrag erzeugt, wobei für das **Control VD** das Symbol VD verwendet wird.



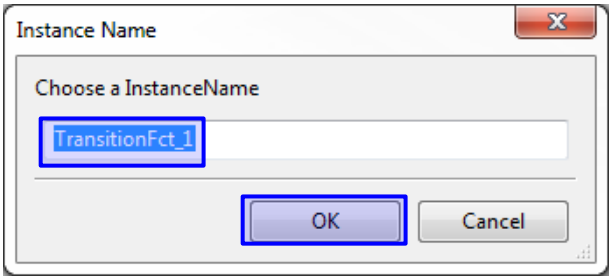
Es ist zu beachten, dass eine Abfrage von `GDI_Status` auf einem **Control VD** nicht erlaubt ist.

Das **Control VD** enthält die zwei **Interfaces**, `"TransitionFct"` (**Transition Function**) und `"DeviceBaseFct"` (**Device Base Function**).

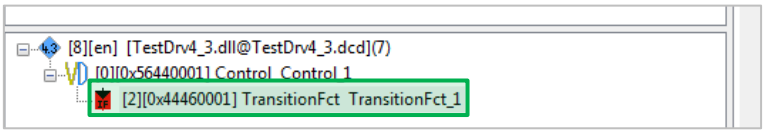
Um **GDI Phasen**-Übergänge ausführen zu können, müssen **Transition Function** entsprechend Kapitel 7.4.2 und die dort enthaltenen **Operations** entsprechend Kapitel 7.4.4 instanziiert werden.



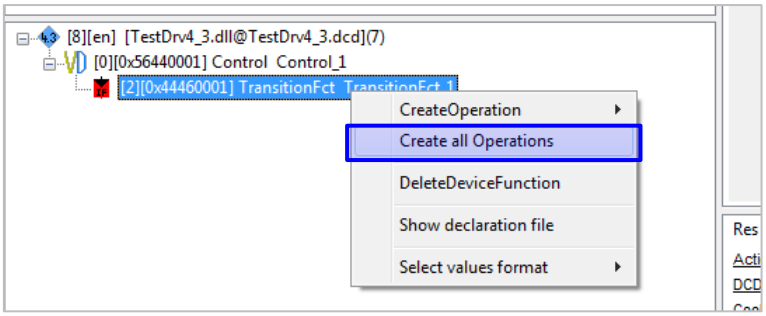
Zum Anlegen von Transition Function wird der Instanz-Name erfragt.



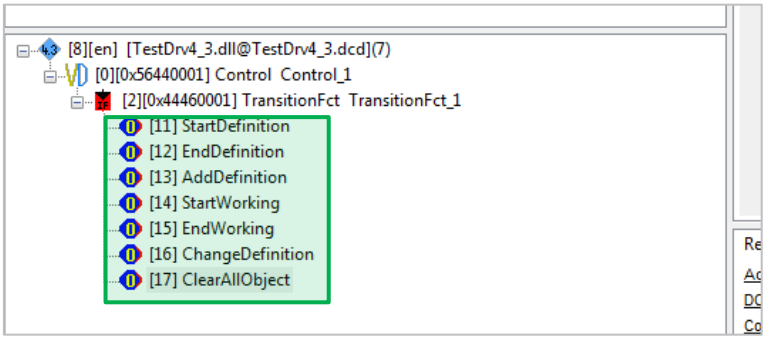
Nach Bestätigung mittels OK wird das **Function Object** mit dem Namen „TransitionFct_1“ angelegt.



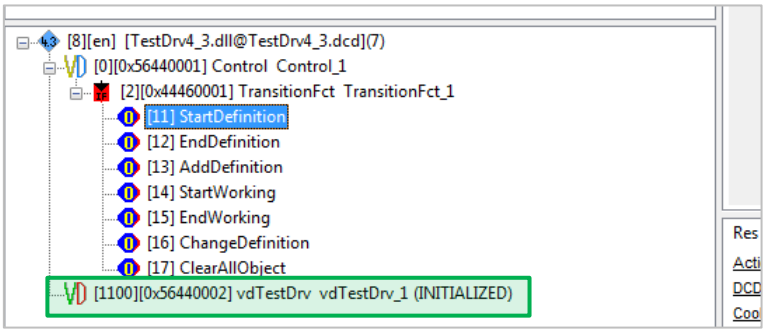
Auf dem **Function Object** können nun beispielsweise alle notwendigen Operationen durch Anwahl der Kontextmenü-Punktes Create all Operations angelegt werden.



Alle so angelegten **Operations** erscheinen als **Operation**-Instanzen im Instanzen-Baum.

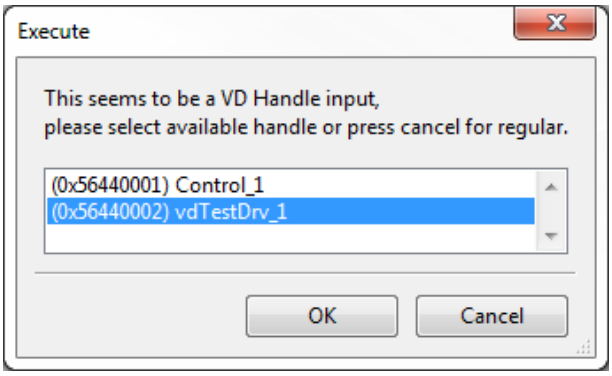


Zur Demonstration des in diesem Kapitel verwendeten Manuellen **Betriebsmodus** wurde das Anlegen eines **Virtual Device** „`vdTestDrv_1`“ entsprechend Kapitel 7.4.1 vorgezogen.



Mit Hilfe dieser **Operations** ist es möglich, die **GDI Phasen** manuell zu schalten. Jeder dieser **Operations** ist bei Ausführung das Handle des **Virtual Device** zu übergeben, für welches der **GDI Phasen**-Übergang durchgeführt werden soll.

Dazu wird ein Auswahldialog eingeblendet, welcher eine Liste der bereits instanziierten **Virtual Devices** mit ihren zugeordneten Handles aufzeigt.



Durch Auswahl von OK wird die entsprechende **Operation** für den **GDI Phasen**-Übergang des ausgewählten **Virtual Device** im übergeordneten **Device Driver** aufgerufen.

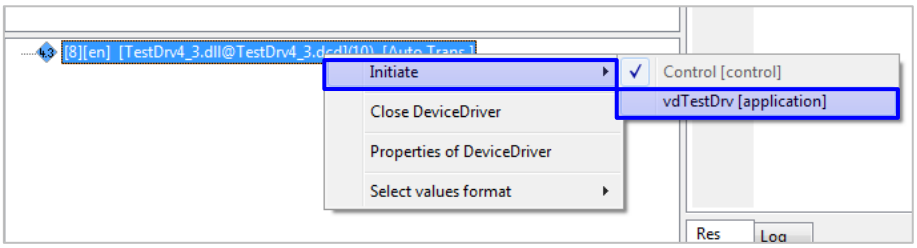
Wird der Dialog mit dem Schalter Cancel abgebrochen, so erscheint der normale Eingabedialog für Eingabewerte von **Operations** [7.5.3]. Damit ist es möglich, zur Ausführung von Negativ-Tests beliebige Werte als VD-Handle an den **Device Driver** zu übergeben.

7.4 Anlegen von Instanzen

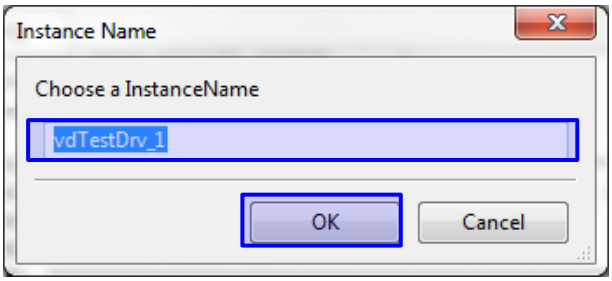
7.4.1 Instanziieren von Virtual Device

Der weitere Ablauf wurde im Automatischen **Betriebsmodus** durchgeführt.

Das **Module** „vdTestDrv“ kann nun mit Hilfe des Kontextmenüpunktes Initiate des Instanzen-Baums [5.6] instanziiert werden.



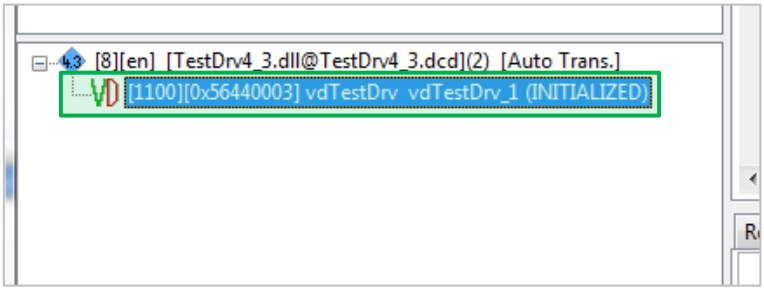
Es erfolgt die Abfrage des Instanz-Namens des so zu erstellenden **Virtual Device**. Ein eindeutiger Vorschlag ist bereits in den erscheinenden Dialog eingetragen, kann aber auch geändert werden. Die vergebenen Namen müssen eindeutig sein.



Der automatische Vorschlag besteht normalerweise aus dem Namen des **Modules** aus der **DCD**, erweitert durch einen Unterstrich und eine laufende Nummer. Die Zählung beginnt bei `1`. Diese Vergabeweise trifft auch auf erzeugte Instanzen von **Function Object** zu. Abweichungen können auftreten, falls der nächste zu vergebende Name bereits existiert (z.B. erzeugt von einem **Makro** oder durch Eingabe eigener Namen im Dialog). Auf jeden Fall wird sichergestellt das der automatische Vorschlag eindeutig ist.

Sollte das zu instanziierende **Module Initiate Parameter** besitzen, so erscheint eine Kombination aus dem Eingabedialog für Daten [5.10] und dem Eingabefeld für den Instanz-Namen. Die **Initiate Parameter** sind auf geeignete Werte zu setzen. Die Behandlung des Instanz-Namens ist identisch.

Nach Bestätigung durch Anwahl des Schalters OK wird im Instanzen-Baum [5.6] der folgende Eintrag erzeugt.

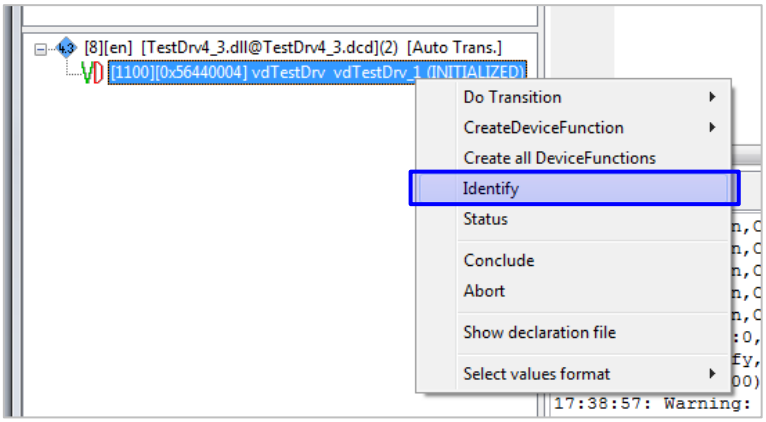


In den runden Klammern hinter der Namensangabe ist die aktuelle **GDI Phase** des **Virtual Device** zu erkennen. Für Informationen, welche **GDI Phasen** es gibt, welche **GDI Phasen**-Übergänge möglich sind und welche Funktionen des **Device Driver** in welcher **GDI Phase** ausgeführt werden dürfen, sei auf [GDI-Doc3] verwiesen.

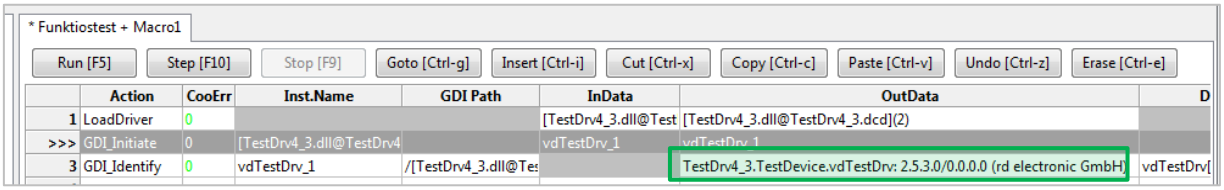
Zusätzlich wird ein neuer Eintrag in der Schrittliste [5.7] erzeugt.

7.4.1.1 Identify

Mit dem Menüpunkt `Identify` des Kontextmenüs einer Instanz eines **Virtual Device** im Instanzen-Baum [5.6] ist es möglich, ein `GDI_Identify`, mit dem entsprechenden **Virtual Device** als Parameter, auszuführen.



Die Funktion liefert ihr Ergebnis in der Spalte `OutData` [5.7.1] der Schrittliste.

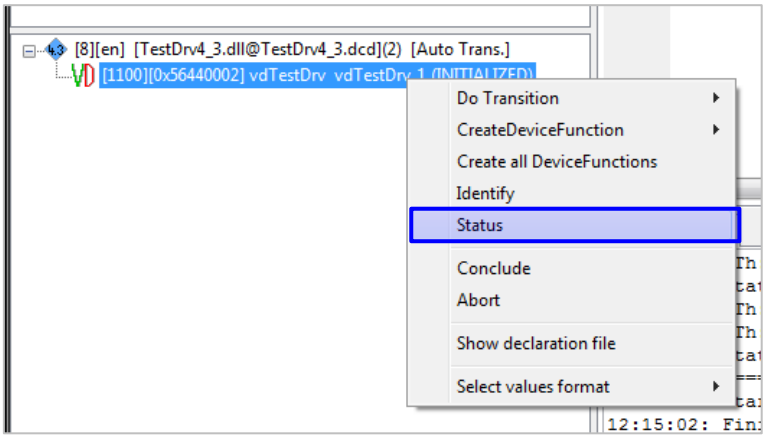


Weiterhin erscheint das Ergebnis des Aufrufs in der Log-Ansicht [5.9].

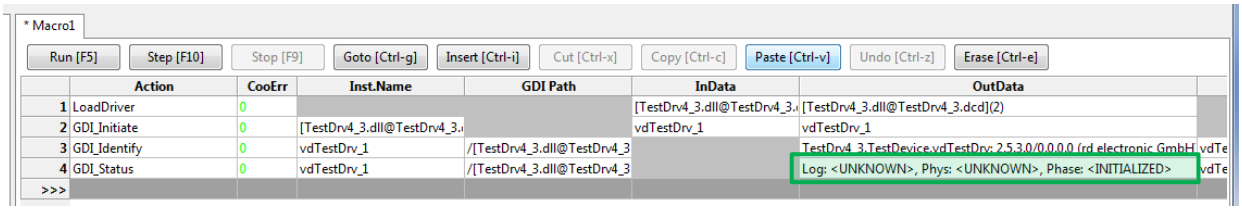
```
17:41:53: Identify, DeviceVersion: 0.0.0.0(0x00000000), DriverName:
TestDrv4_3.TestDevice.vdTestDrv, DriverVersion: 2.5.3.0(0x02050300), Factory:
rd electronic GmbH
```

7.4.1.2 Status

Mit dem Menüpunkt `Status` des Kontextmenüs einer Instanz eines **Virtual Device** im Instanzen-Baum [5.6] ist es möglich den Status (`GDI_Status`) des entsprechenden **Virtual Device** abzufragen.



Die Funktion liefert ihr Ergebnis in der Spalte `OutData` [5.7.1] der Schrittliste.



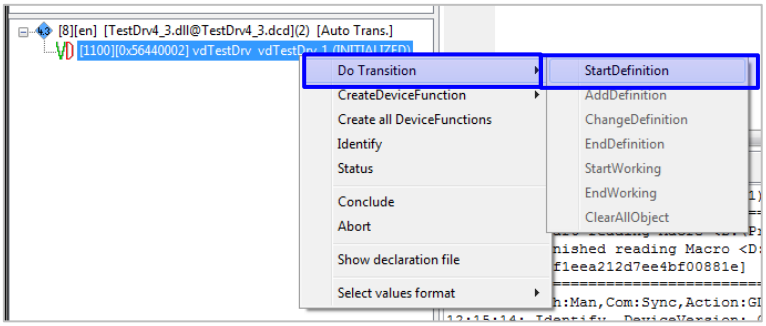
Weiterhin erscheint das Ergebnis des Aufrufs in der Log-Ansicht [5.9].

```
12:15:36: Status, Log: <UNKNOWN>(-1), Phys: <UNKNOWN>(-1), Phase:
<INITIALIZED>(1), Qual: 0, Grade: 0, Code: 0, Text:
```

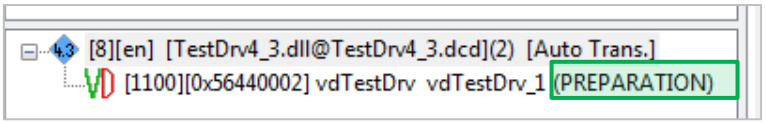
7.4.1.3 Transition

Im Automatischen **Betriebsmodus** [5.5.2.6] werden die **GDI Phasen** von der integrierten **Coordinator Engine** automatisch geschaltet. Zu diesem Zweck erzeugt die **Coordinator Engine** intern das dafür nötige **Control VD** und darin **Transition Function**. Die **Operations** von **Transition Function** dienen zum Schalten der **GDI Phasen**.

Es ist jedoch möglich, und in besonderen Fällen auch erforderlich, die **GDI Phasen**-Übergänge manuell herbeiführen zu können. Dies kann ebenfalls mit dem unten abgebildeten Kontextmenü erfolgen.



Ein erfolgreicher **GDI Phasen**-Übergang bewirkt die Änderung der Anzeige der aktuellen **GDI Phase** der entsprechenden Instanz des **Virtual Device** im Instanzen-Baum [5.6].

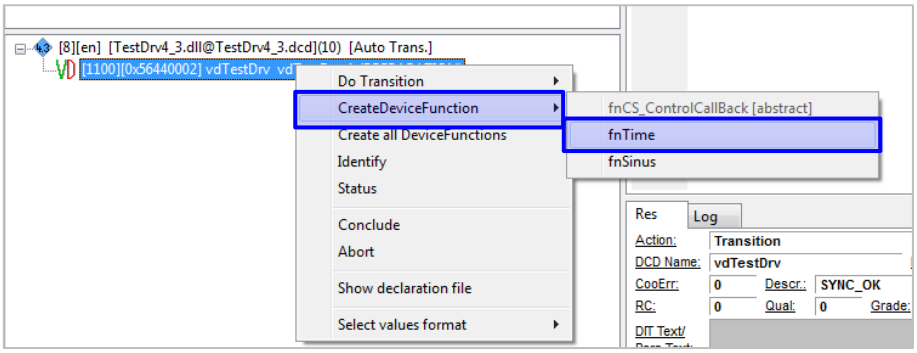


7.4.2 Instanziieren von Function Objects

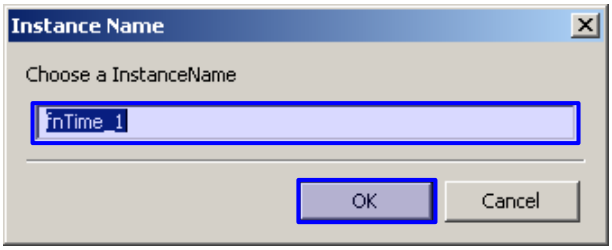
Der Vorgang des Instanziierens eines **Function Object** ist ähnlich dem des Instanziierens eines **Virtual Device**. Der zugehörige Kontextmenüpunkt CreateDeviceFunction befindet sich jedoch auf dem entsprechenden Eintrag des **Virtual Device** im Instanzen-Baum [5.6].

Es gibt zwei verschiedene Möglichkeiten ein **Function Object** zu instanziieren. Zum einen kann im Kontextmenü eines der für dieses **Virtual Device** verfügbaren **Interfaces** zum Instanziieren ausgewählt werden. Zum anderen ist es auch möglich, alle **Interfaces** eines **Virtual Device** hintereinander instanziieren zu lassen. Es ist dabei jedoch zu beachten, dass auch versucht wird, nicht erlaubte **Interfaces** (z.B. **Abstract Inerfaces**) zu instanziieren und alle mehrfach instanziierbaren **Interfaces** nur einmal abgearbeitet werden.

Es ist zu beachten, dass an der Oberfläche des Programms der alternative Begriff **Device Function** [GDI-Doc3] für **Interfaces** verwendet wird.

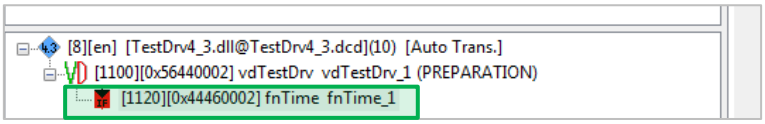


Daraufhin erfolgt die Abfrage des Instanz-Namens für das zu erstellende **Function Object**. Das Handling ist identisch mit der Vergabe des Instanz-Namens für ein **Virtual Device**.



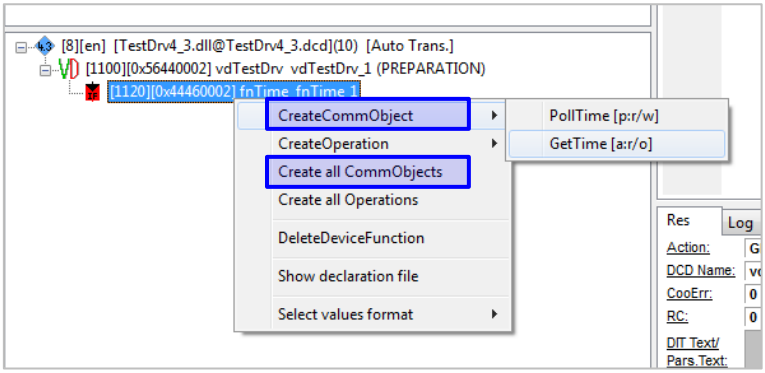
Sollte das zu instanziierende **Interface Create Parameters** besitzen, so erscheint eine Kombination aus dem Eingabedialog für Daten [5.10] und dem Eingabefeld für den Instanz-Namen. Die **Create Parameters** sind auf geeignete Werte zu setzen. Die Behandlung des Instanz-Namens ist identisch.

Das instanziierte **Function Object** erscheint als Untereintrag des **Virtual Device** im Instanzen-Baum [5.6].

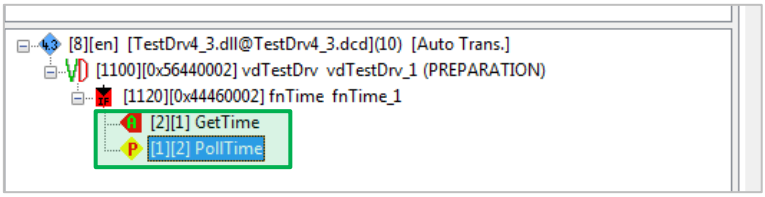


7.4.3 Instanziieren von Communication Objects

Communication Objects können einzeln durch Anwahl von `CreateComObject` oder gesamtheitlich durch Anwahl von `Create all CommObjects` aus dem Kontextmenü über dem **Function Object** instanziiert werden.



Instanziierte **Communication Objects** erscheinen nun als Untereinträge des **Function Object**.

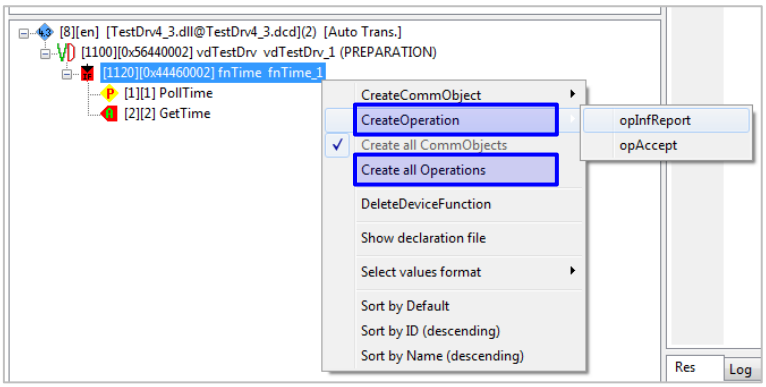


7.4.4 Instanziieren von Operations

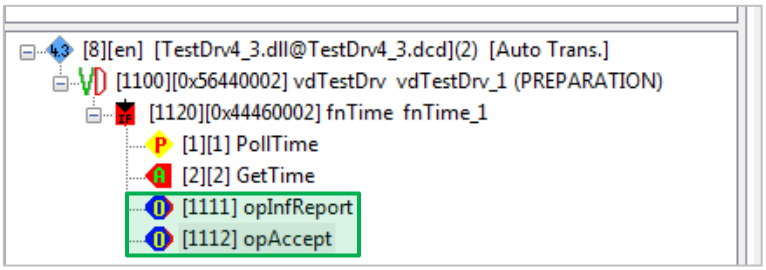
Im **GDI** Standard ist es normalerweise nicht notwendig, **Operations** im **Device Driver** vor ihrer Benutzung zu instanziieren. Sie sind an der API des **Device Driver** verfügbar sobald das zugehörige **Function Object** instanziiert worden ist. Die objektorientierte Schnittstelle der **Coordinator Engine** erzwingt jedoch die Erzeugung zugehöriger **Coordinator Engine** Objekte. Dies wird bis in die Anwendung abgebildet.

Aktionsnamen für Eintragungen in der Schrittliste [5.7] zum Instanziieren von **Operations** ist nicht der Prefix „GDI_“ vorangestellt, da keine Funktionsaufrufe in die zugeordneten **Device Drivers** erfolgen.

Operations können einzeln durch Anwahl von CreateOperation oder gesamtheitlich durch Anwahl von Create all Operations aus dem Kontextmenü über dem **Function Object** instanziiert werden.



Instanziierte **Operations** erscheinen nun als Untereinträge des **Function Object**.



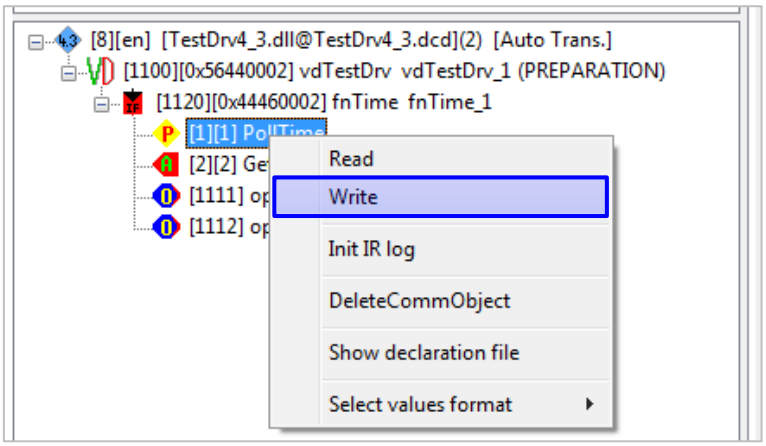
Operations können nicht explizit gelöscht werden, vielmehr erfolgt dies mit Abbau des übergeordneten **Function Object**.

7.5 Arbeiten mit Instanzen

7.5.1 Schreiben auf einem Communication Object

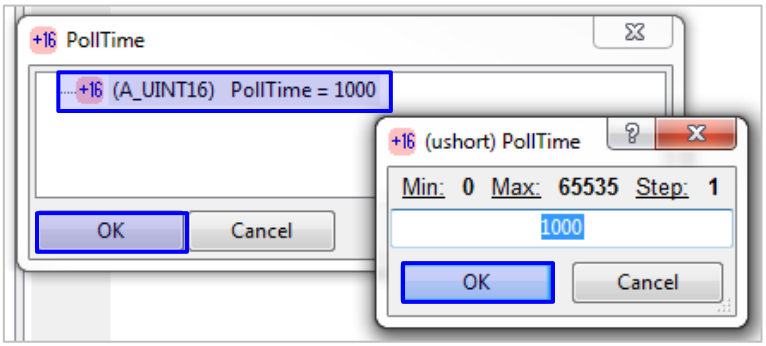
Mit den in den vorangegangenen Abschnitten instanziierten **Communication Objects** ist es möglich, Werte in den **Device Driver** zu schreiben. Aus Sicht der Anwendung macht es keinen Unterschied, ob es sich dabei um einen **Parameter** oder ein **Attribute** handelt. Sie unterscheiden sich jedoch hinsichtlich der **GDI Phasen**, in denen das Schreiben erlaubt ist.

Das Schreiben wird durch den Kontextmenüpunkt Write über dem **Communication Object** ermöglicht.



Das Schreiben ist nur möglich, wenn das entsprechende **Communication Object** nicht readonly. Erkennbar ist dies am zugehörigen Symbol [5.6.1] der Instanz.

In den darauf folgenden Dialogen können nun die zu übertragenden Werte eingestellt werden [5.10.3].



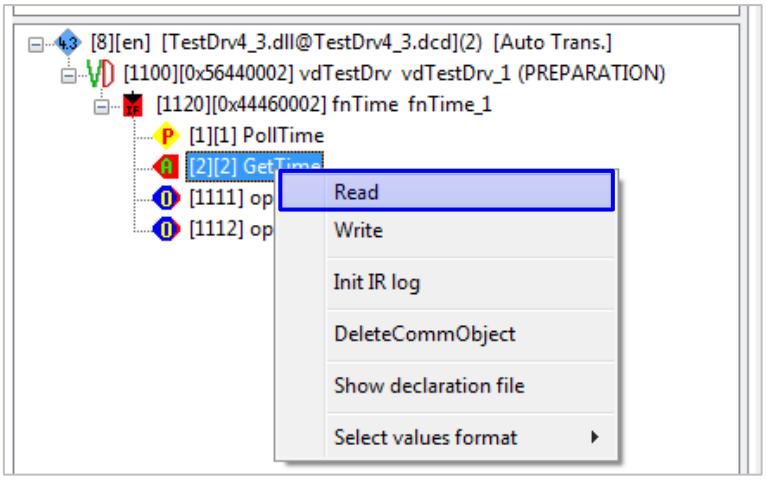
Nach Bestätigung der Dialoge wird der **Device Driver** Funktion `GDI_Write` aufgerufen und der neue Wert an den **Device Driver** übertragen.

Im Beispiel wurde das **Attribut** `PollTime` auf den Wert `1000` gesetzt. Im Falle der Aktivierung von **Information Reports** auf dem **Communication Object** `GetTime` würden **Information Reports** alle 1000 Millisekunden an **GINA2010** übertragen werden.

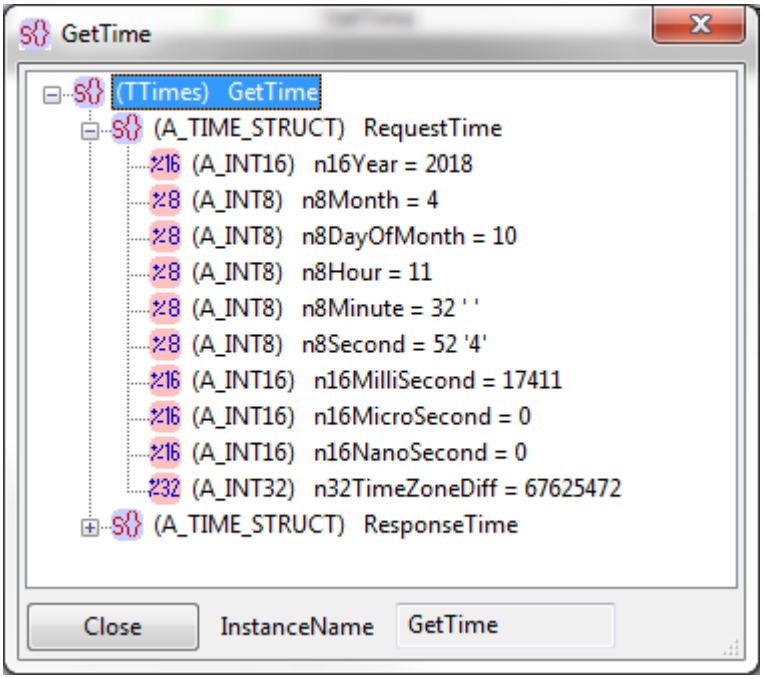
7.5.2 Lesen auf einem Communication Object

Mit den in den vorangegangenen Abschnitten instanziierten **Communication Objects** ist es möglich, Werte aus dem **Device Driver** zu lesen. Aus Sicht der Anwendung macht es keinen Unterschied ob es sich dabei um einen **Parameter** oder ein **Attribute** handelt.

Das Lesen wird durch den Kontextmenüpunkt `Read` über dem **Communication Object** ermöglicht.



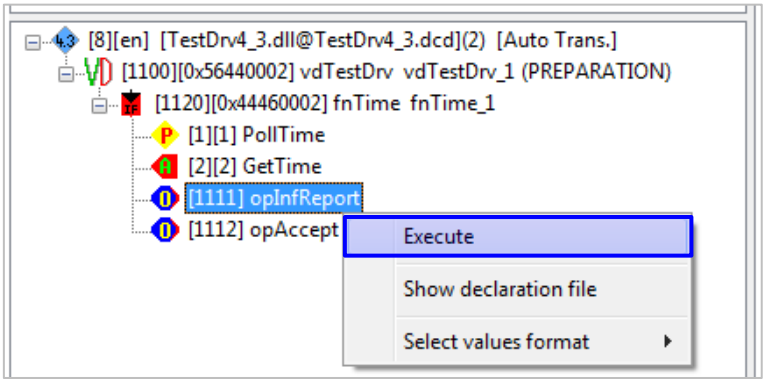
Im darauffolgenden Dialog werden die aus dem **Device Driver** gelesenen Werte dargestellt.



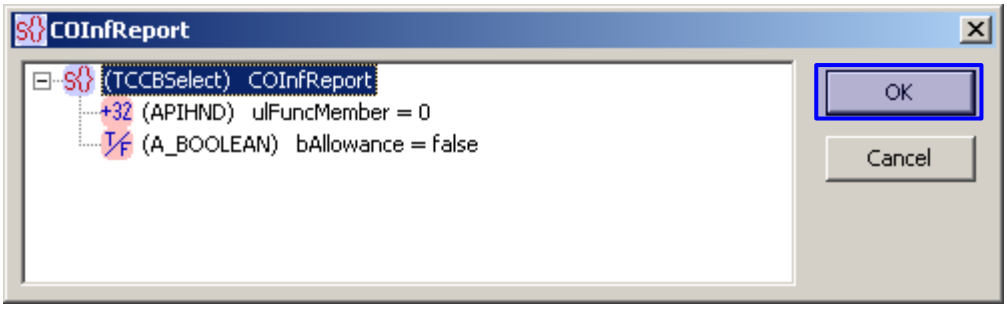
7.5.3 Ausführen einer Operation

Die Benutzung von **Operations** ist ähnlich dem Schreiben [7.5.1] und Lesen [7.5.2] von **Communication Objects**.

Das Ausführen wird durch den Kontextmenüpunkt `Execute` über der **Operation** ermöglicht.



Wenn die **Operation** Eingabeparameter hat (siehe Symbol [5.6.1]), wird ein Eingabe-Dialog [5.10] eingeblendet.



Durch Bestätigen des Dialogs durch Auswahl des Schalters `OK` wird die **Operation** mit den eingetragenen Eingabeparametern im zugeordneten **Device Driver** ausgeführt.

Im Beispiel werden vorerst die **Information Reports** auf allen **Communcation Objects** (`ulFuncMember=0`) des **Function Objects** deaktiviert (`bAllowance=false`).

Verfügt die **Operation** über Ausgabeparameter (siehe Symbol [5.6.1]), so werden diese in einem Dialog analog Kapitel 7.5.2 dargestellt.

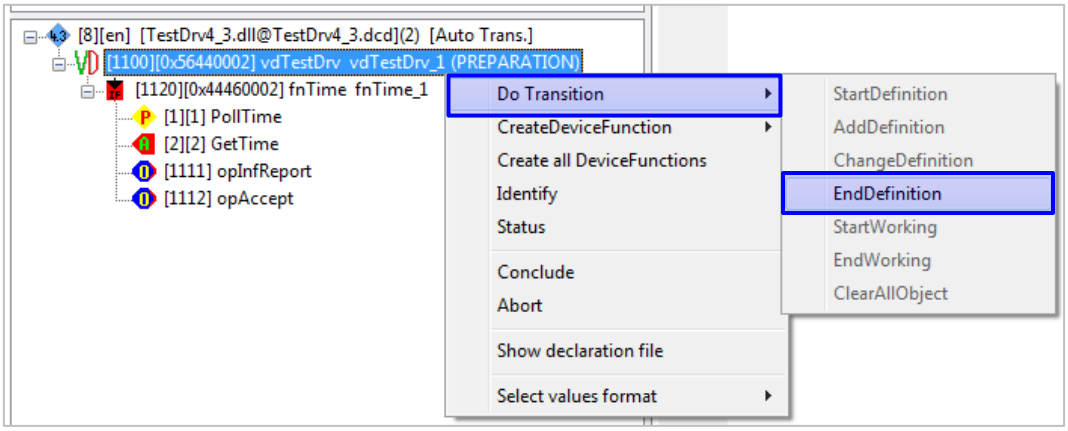
7.5.4 Information Reports

Das Beispiel zeigt im weiteren Verlauf, die notwendigen Schaltungen der GDI Phasen, die Aktivierung und die Anzeige von **Information Reports** aus dem **RDE Testdriver**.

7.5.4.1 Schalten der GDI Phase

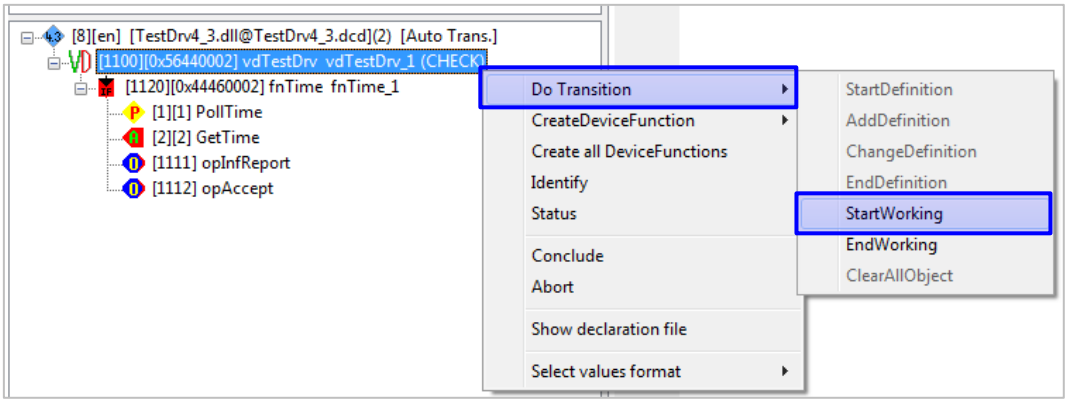
Information Reports können nur in den **GDI Phasen** [GDI-Doc3] `Working` oder `Revise` aktiviert werden.

Es wird daher mittels Auswahl von `EndDefinition` in die **GDI Phase** `Check` geschaltet.

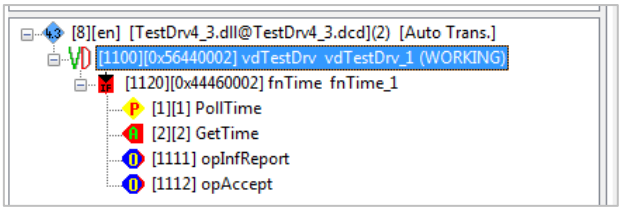


Die erreichte **GDI Phase** `Check` wird hinter dem **Function Object** angezeigt.

Danach wird mittels Auswahl von `StartWorking` in die **GDI Phase** `Working` geschaltet.



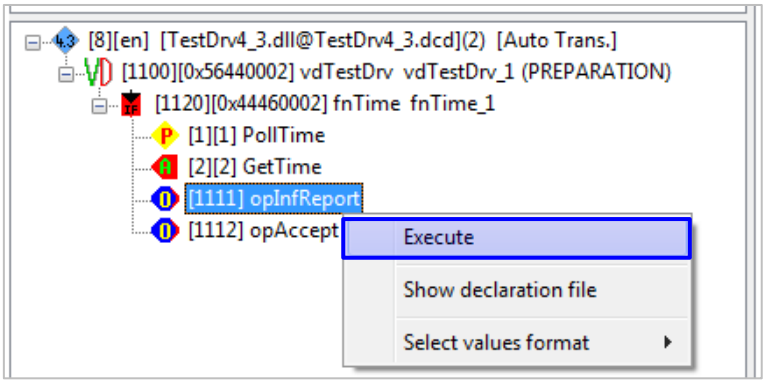
Die erreichte **GDI Phase** `Working` wird hinter dem **Function Object** angezeigt.



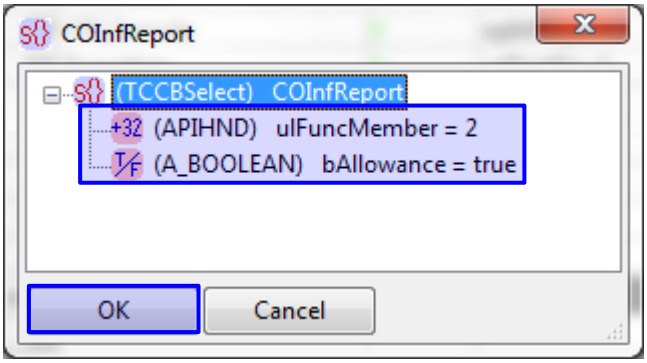
In diesem Zustand ist es nun erlaubt, **Information Reports** zu aktivieren.

7.5.4.2 Aktivieren der Information Reports

Analog Kapitel 7.5.3 wird die **Operation** `opInfReport` ausgeführt.



Im Beispiel wird der Parameter `ulFuncMember` auf `2` und der Parameter `bAllowance` auf `true` gesetzt. Damit werden **Information Reports** nur auf dem **Communication Object** „`GetTime`“ (`ID=2`) aktiviert.

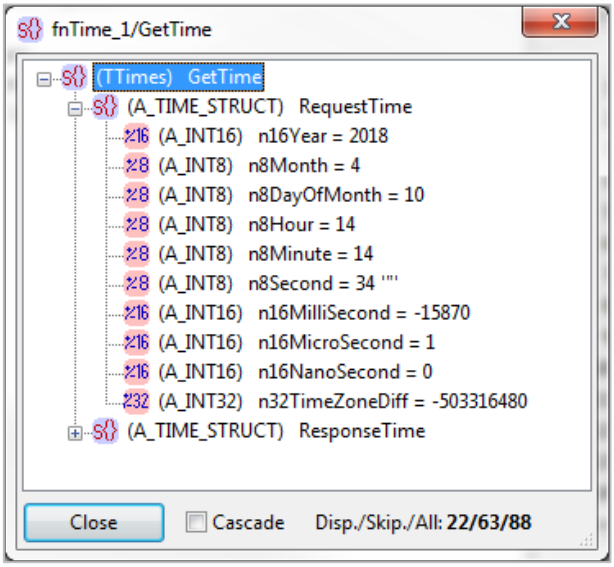


Nach Bestätigen des Dialogs mittels OK werden die **Information Reports** auf dem **Communication Object** „GetTime“ aktiviert und der **RDE Testdriver** beginnt entsprechend seinen Einstellungen mit der Übertragung der **Information Reports**.

7.5.4.3 Anzeige der Information Reports

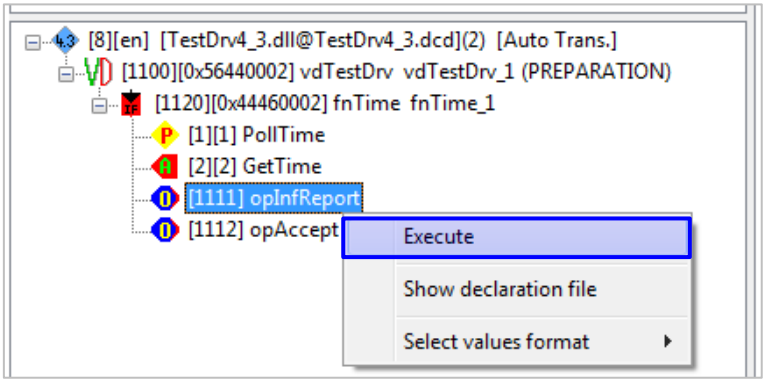
Nach erfolgreicher Aktivierung der **Information Reports** sendet der **RDE Testdriver** jede Sekunde einen **Information Report** [5.11] an **GINA2010**.

Die so zyklisch an **GINA2010** übertragenen Werte werden in gleicher Weise dargestellt, wie beim Lesen von **Communication Objects** [7.5.2].

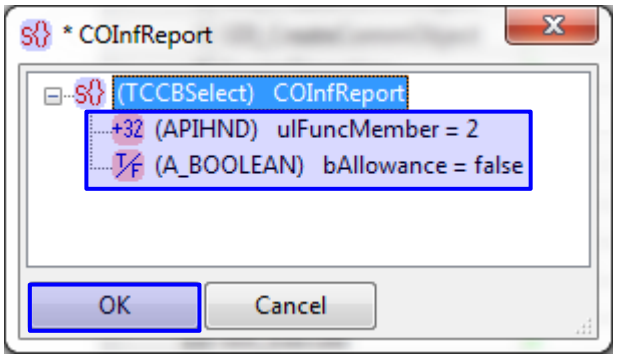


7.5.4.4 Deaktivieren von Information Reports

Analog Kapitel 7.5.3 wird die **Operation** `opInfReport` ausgeführt.



Im Beispiel wird der Parameter `ulFuncMember` auf `2` und der Parameter `bAllowance` auf `false` gesetzt. Damit werden **Information Reports** nur auf dem **Communication Object** „`GetTime`“ (`ID=2`) deaktiviert.

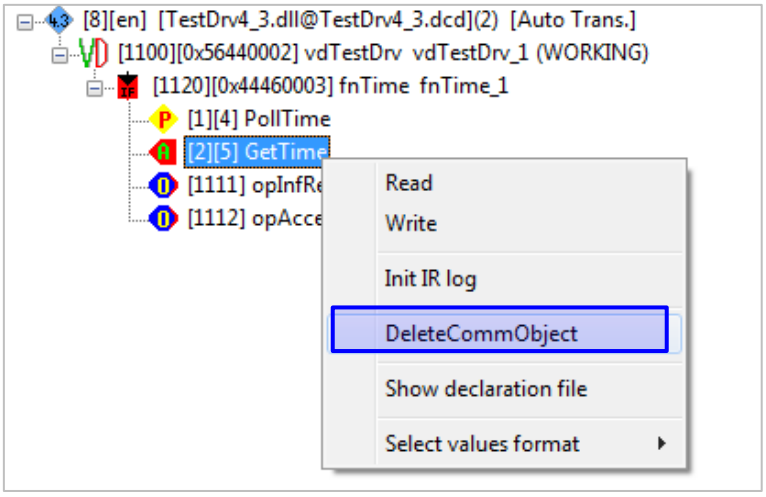


Nach Bestätigen des Dialogs mittels `OK` werden die **Information Reports** auf dem **Communication Object** „`GetTime`“ deaktiviert und der **RDE Testdriver** überträgt keine **Information Reports** mehr an **GINA2010**.

7.6 Abbauen von Instanzen

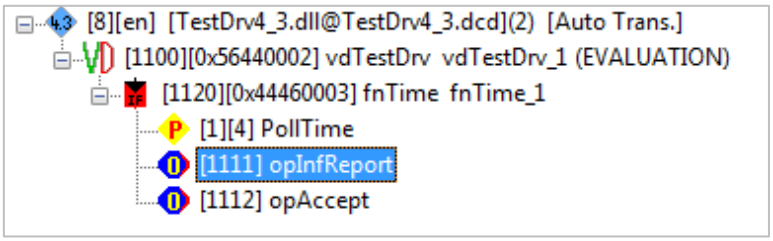
7.6.1 Löschen von Communication Objects

Zum Abbau instanziierter **Communication Objects** bieten die Kontextmenüs über diesen Instanzen die Aktion `DeleteCommObject` an.



Durch Ausführen dieser Aktion schaltet die integrierte Coordinator Engine im Falle des Automatischen **Betriebsmodus** in die **GDI Phase** `Evaluation`., da das Löschen von **Communication Objects** in der **GDI Phase** `Working` nicht erlaubt ist.

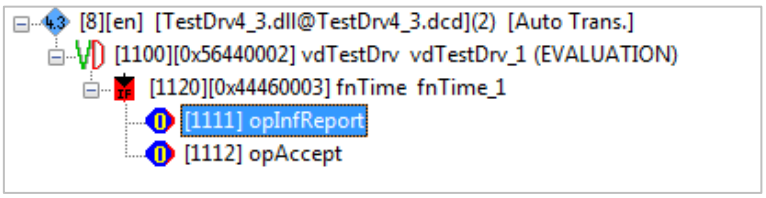
Im Falle des Manuellen **Betriebsmodus** muss der Anwender die Schaltung der **GDI Phase** selbst analog Kapitel 7.5.4.1 durch Aufruf der **GDI Phasen Operation** `EndWorking` vornehmen, bevor er die Aktion `DeleteCommObject` ausführt.



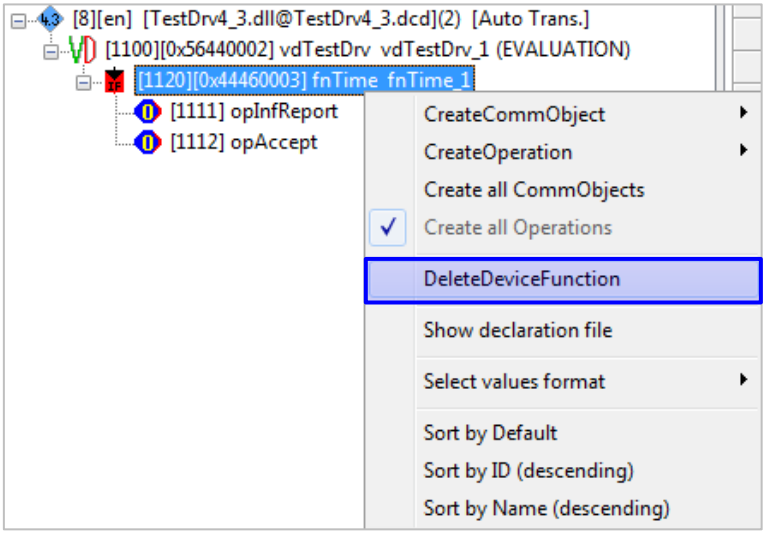
Der Aufruf an den **Device Driver** erfolgt. Nach erfolgreichem Aufruf wird das **Communication Object** aus dem Instanzen-Baum entfernt.

7.6.2 Löschen von Function Objects

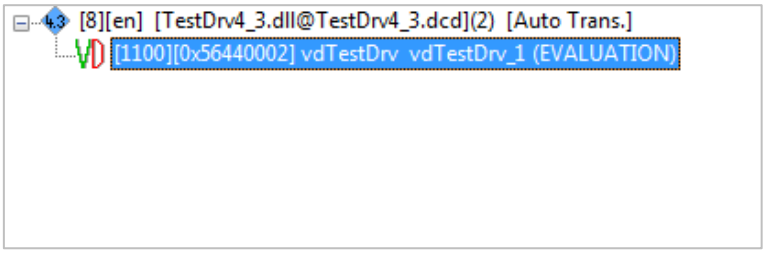
Das Löschen einer **Device Function** ist nur dann erlaubt, wenn keine **Communication Objects** mehr enthalten sind. Analog Kapitel 7.6.1 ist somit auch die Instanz `"PollTime"` zu löschen, bevor mit den Löschen des **Function Object** fortgefahren werden kann.



Durch Aufruf der Aktion `DeleteDeviceFunction` aus dem Kontextmenü für **Device Functions** ist es nun möglich, die Instanz `„fnTime_1"` zu löschen.

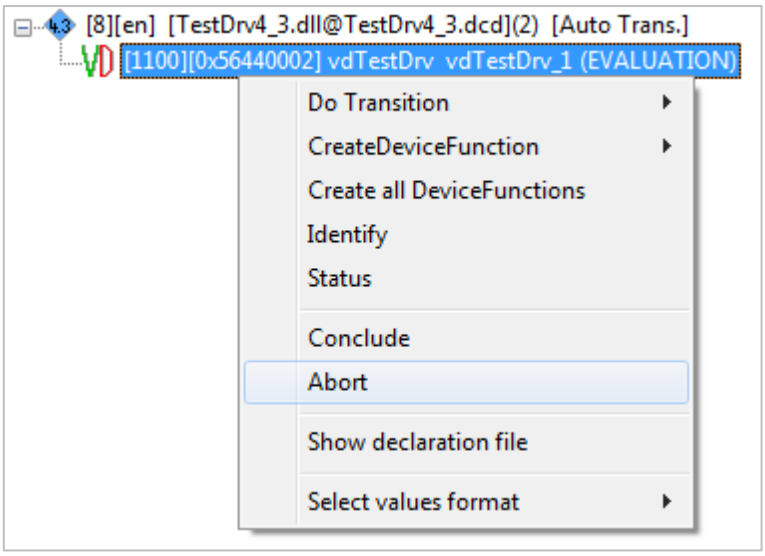


Der Aufruf an den **Device Driver** erfolgt. Nach erfolgreichem Aufruf wird die **Devcice Function** `„fnTime_1"` aus dem Instanzen-Baum entfernt.

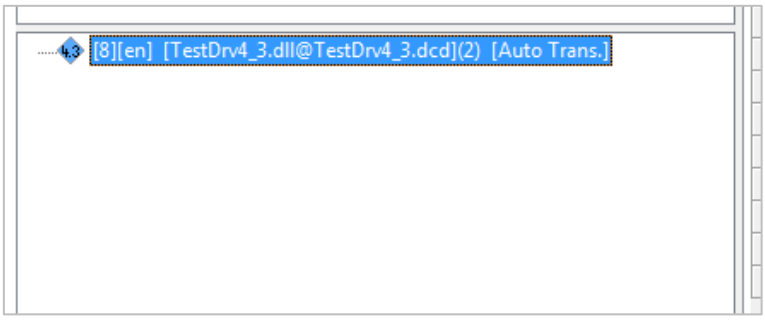


7.6.3 Löschen von Virtual Devices

Durch Aufruf der Aktion `Conclude` aus dem Kontextmenü für **Virtual Devices** ist es nun möglich, die Instanz „vdTestDrv_1“ zu löschen.

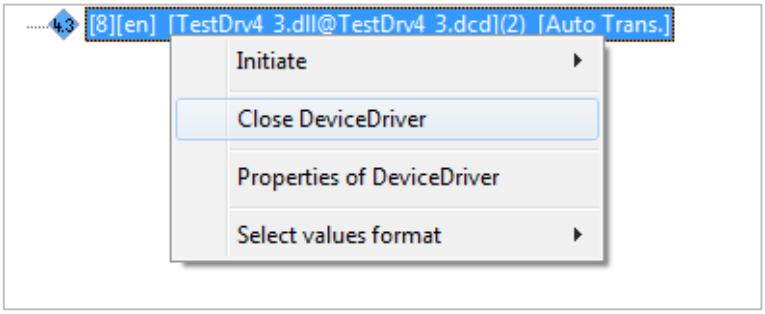


Der Aufruf an den **Device Driver** erfolgt. Nach erfolgreichem Aufruf wird das **Virtual Device** „vdTestDrv_1“ aus dem Instanzen-Baum entfernt.



7.7 Entladen des Device Driver

Durch Aufruf der Aktion Close DeviceDriver aus dem Kontextmenü für **Device Drivers** ist es nun möglich, die **RDE Testdriver** zu entladen.



Nach erfolgreicher Entladung wird **RDE Testdriver** aus dem Instanzen-Baum entfernt.