

DIPLOMARBEIT

**Online Programmierung von Robotersystemen gemäß der
OPEN ROBOT INTERFACE (ORI) - ARCHITECTURE**

ausgeführt am Institut für Fertigungstechnik
der Technischen Universität Wien

unter der Anleitung von O.Univ.Prof. Dipl.-Ing. Dr.techn. Weseslindtner Helmar
und Univ.Ass. Dipl.-Ing. Stopper Markus
als verantwortlich mitwirkenden Universitätsassistenten

durch

Robert Panzirsch
Herklotzgasse 7/17
1150 Wien

Wien, 10.03.1999

Danksagung

Mein Dank gilt Herrn O.Univ.Prof. Dipl.-Ing. Dr.techn. Weseslindtner Helmar, daß er diese Arbeit ermöglicht hat, meinem Betreuer Univ.Ass. Dipl.-Ing. Markus Stopper für zahlreiche Anregungen und das Beisteuern zusätzlicher Informationen, Hrn. Dipl.-Ing. Bernhard Angerer für seine ausgezeichnete Hilfe bei der Implementierung in Delphi sowie meiner Freundin Belinda Zottl für ihre Geduld und Unterstützung beim Studium.

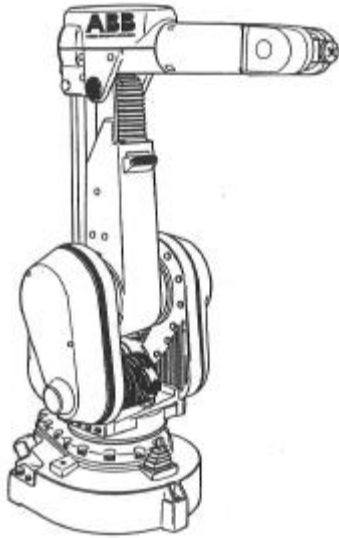
Inhaltsverzeichnis

1. EINLEITUNG	5
2. OBJEKTORIENTIERTE PROGRAMMIERUNG (OOP)	6
2.1. Einleitung.....	6
2.2. Kapselung	6
2.3. Vererbung	7
2.4. Polymorphismus	8
2.5. Aggregation.....	9
3. OOP IN DELPHI.....	10
3.1. Einleitung.....	10
3.2. PROPERTIES.....	10
3.3. METHODEN.....	11
3.4. EVENTS	12
4. ORI IM ENTWICKLUNGSSYSTEM DELPHI	13
4.1. Einleitung.....	13
4.2. OPEN ROBOT INTERFACE (ORI) - ARCHITECTURE.....	13
4.3. DELPHIS IDE	13
4.4. Komponenten in der IDE	14
5. SCHICHTENTRENNUNG UND ORI - KOMPONENTEN.....	15
5.1. Einleitung.....	15
5.2. ROBOT PHYSICAL LAYER (RPL)	16
5.3. ROBOT COMMUNICATION LAYER (RCL)	16
5.4. ROBOT FUNCTION LAYER (RFL).....	17
5.5. ROBOT APPLICATION LAYER (RAL).....	18
5.6. ORI - Komponenten	18
6. ROBOTER HARDWARE	19
6.1. Einleitung.....	19
6.2. Mechanischer Teil.....	19
6.3. Arbeitsbereich	20
6.4. Koordinatensysteme	20
6.4.1. <i>Rechtwinkeliges sockelorientiertes Koordinatensystem</i>	20
6.4.2. <i>Rechtwinkeliges handgelenkorientiertes Koordinatensystem</i>	22
6.4.3. <i>Rechtwinkeliges werkzeugorientiertes Koordinatensystem</i>	22
6.4.4. <i>Roboterachsenkoordinatensystem</i>	23
6.5. Steuerung	24
6.6. PROGRAMMING UNIT	25

6.7. Schnittstellen	26
7. ROBOTER PROTOKOLLE	27
7.1. Einleitung.....	27
7.2. RS232	27
7.2.1. Hardwareverbindung	27
7.2.2. Konfiguration.....	29
7.3. ADLP10.....	30
7.3.1. Einleitung	30
7.3.2. Steuerzeichen	30
7.3.3. Kommunikationsphasen.....	31
7.3.4. Sicherungsmechanismen.....	33
7.3.5. Kommunikations-Beispiele	34
7.4. ARAP	38
7.4.1. Einleitung	38
7.4.2. Telegrammaufbau	38
7.4.3. Telegrammtypen	41
7.4.4. FUNCTION CODES	42
8. IMPLEMENTIERUNG DER ORI - KOMPONENTEN.....	48
8.1. Einleitung.....	48
8.2. RS232	48
8.2.1. Konfiguration.....	48
8.2.2. Schnittstelle zu ADLP10.....	49
8.2.3. Der Timer	49
8.3. ADLP10.....	50
8.3.1. Konfiguration.....	50
8.3.2. Schnittstelle zu ARAP.....	50
8.4. ARAP	50
8.4.1. Konfiguration.....	51
8.4.2. Schnittstelle zu Anwendungen bzw. RAL	51
8.5. Spezielle Datenstrukturen	58
8.5.1. TRPOINT.....	58
8.6. EXCEPTIONS.....	59
9. BEISPIEL – ANWENDUNG ORIDEMO	61
9.1. Einleitung.....	61
9.2. Oberfläche.....	61
9.3. Programmierdetails	63
9.3.1. Aufruf einer RAL – Funktion.....	63

9.3.2.	<i>TEACHING und MESSAGES</i>	64
9.3.3.	<i>BUSY</i>	65
10.	SCHLUßWORT	66
11.	ANHANG	67
11.1.	FUNCTION CODES.....	67
11.1.1.	<i>FUNCTION CODE 1: Roboterprogramm senden</i>	67
11.1.2.	<i>FUNCTION CODE 2: Programm starten</i>	69
11.1.3.	<i>FUNCTION CODE 3: Programm stoppen</i>	70
11.1.4.	<i>FUNCTION CODE 4: TCP Register lesen</i>	71
11.1.5.	<i>FUNCTION CODE 5: LOCATION Register lesen</i>	72
11.1.6.	<i>FUNCTION CODE 6: Register DATA lesen</i>	74
11.1.7.	<i>FUNCTION CODE 7: SENSOR Register lesen</i>	75
11.1.8.	<i>FUNCTION CODE 8: DIGITAL INPUTS lesen</i>	76
11.1.9.	<i>FUNCTION CODE 9: DIGITAL OUTPUTS lesen</i>	78
11.1.10.	<i>FUNCTION CODE 10: CONFIGURATION DATA lesen</i>	79
11.1.11.	<i>FUNCTION CODE 11: FRAME Register lesen</i>	81
11.1.12.	<i>FUNCTION CODE 12: TCP Register schreiben</i>	82
11.1.13.	<i>FUNCTION CODE 13: LOCATION Register schreiben</i>	83
11.1.14.	<i>FUNCTION CODE 14: Register DATA schreiben</i>	85
11.1.15.	<i>FUNCTION CODE 15: SENSOR Register schreiben</i>	86
11.1.16.	<i>FUNCTION CODE 16: DIGITAL OUTPUTS schreiben</i>	88
11.1.17.	<i>FUNCTION CODE 17: CONFIGURATION DATA schreiben</i>	89
11.1.18.	<i>FUNCTION CODE 18: FRAME Register schreiben</i>	90
11.1.19.	<i>FUNCTION CODE 19: STATUS lesen</i>	92
11.1.20.	<i>FUNCTION CODE 20: MODE schreiben</i>	95
11.1.21.	<i>FUNCTION CODE 21: PROGRAM STATUS lesen</i>	97
11.1.22.	<i>FUNCTION CODE 22: Programm löschen</i>	99
11.1.23.	<i>FUNCTION CODE 23: Programm von Diskette laden</i>	100
11.1.24.	<i>FUNCTION CODE 24: Bewegung</i>	101
11.1.25.	<i>FUNCTION CODE 29: Roboter Programm empfangen</i>	105
11.1.26.	<i>FUNCTION CODE 72: TEACHING</i>	106
11.1.27.	<i>FUNCTION CODE 127: SPONTANEOUS MESSAGE</i>	114
11.2.	Beispiel – Anwendung ORIDEMO, Listing.....	119
11.3.	Literaturverzeichnis	135
11.4.	ASCII - CODE Tabelle	136

1. Einleitung



Die Programmierung von Robotern war bisher nur in sehr Maschinen - oder Firmen - spezifischer Weise und ohne allgemeine Standards möglich. Objektorientierte Programmierung und Komponententechnologie hat es auf diesem Gebiet bisher kaum gegeben. Anwender waren oft gezwungen, mit Firmen – spezifischen Tools zu arbeiten. Sichere, strukturierte, objektorientierte oder grafische Programmierung war daher nicht möglich. Auch die Wiederverwendbarkeit von einmal programmierten Modulen war nur sehr eingeschränkt möglich. Die in dieser Arbeit vorgestellte Implementierung der OPEN ROBOT INTERFACE (ORI) – ARCHITECTURE [STOPPER] bietet die Möglichkeit, auf höchstem Programmierniveau Anwendungen zur Steuerung von Robotern zu schreiben. ORI selbst ist keine Anwendung, sondern eine Architektur, die einerseits ein Interface mit Standard - Roboterfunktionen bietet und andererseits durch den modularen Aufbau universell für jeden Roboter einsetzbar ist. Die ORI – Architektur ist in Schichten gegliedert, die über klar definierte Schnittstellen miteinander kommunizieren. Es ist daher problemlos möglich, Schichten zu tauschen, solange das Interface sich nicht ändert. Die Implementierung erfolgte in DELPHI. In dieser Entwicklungsumgebung sind grafische Programmierung, Objektorientierung und Debugging hervorragend unterstützt.

Die Architektur ist prinzipiell nicht auf Roboter beschränkt, sondern kann für jede Art von Maschine verwendet werden. Die derzeit implementierte Version von ORI beschränkt sich allerdings auf die Steuerung von Robotern.

Der Inhalt der Arbeit beginnt mit theoretischen Begriffen zu Objektorientierung und wie diese in DELPHI anzuwenden sind. DELPHIS Entwicklungssystem und die Einbindung von ORI sind im folgenden Kapitel erläutert. Danach wird auf die ORI – Komponenten eingegangen und die Schichten erklärt. Nach einer Einführung in die Roboter - Hardware des ABB IRB 2000 folgen die Roboter – Protokolle. Es wird die serielle Schnittstelle, das ADLP10 - und das ARAP – Protokoll erläutert. Auch die Implementierung dieser Protokolle in ORI wird erklärt. Zuletzt wird die Beispiel – Anwendung ORIDEMO vorgestellt, welche die Verwendung von ORI vorführt. Im Anhang sind alle implementierten Roboter – Funktionen und die Beispiel – Anwendung abgedruckt.

2. Objektorientierte Programmierung (OOP)

2.1. Einleitung

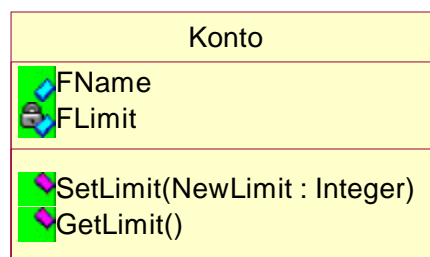
Objektorientierte Programmierung hat klare Vorteile gegenüber einer ausschließlich strukturierten Programmierweise. Wichtigste Eigenschaften sind Daten – Kapselung, Vererbung und Polymorphismus.

Dieses Kapitel widmet sich den prinzipiellen Gedanken objektorientierter Programmierung, während das Folgende sich mit spezielleren Ausprägungen in DELPHI befaßt.

2.2. Kapselung

Daten – Kapselung ist eine der wesentlichen Eigenschaften objektorientierter Programmierung. Objektattribute werden von der Außenwelt versteckt – der Zugriff kann nur über Prozeduren oder Funktionen erfolgen. Durch dieses Kapselung werden unberechtigte Zugriffe unmöglich gemacht. Seiteneffekte, wie sie beim Verwenden von globalen Variablen auftreten, können vermieden werden. Kapselung wird auch oft als INFORMATION HIDING bezeichnet.

Dazu das Beispiel eines Bankkontos:



Die Klasse KONTO hat das private Attribut FLIMIT:

```
TYPE
    Konto = Class
PRIVATE
    FLimit: Integer;
PUBLIC
    FName: String;
```

```
PROCEDURE SetLimit(NewLimit: Integer);  
  
PROCEDURE GetLimit();  
  
END;
```

Dieses Attribut kann nur über die Methode SETLIMIT gesetzt und über die Prozedur GETLIMIT gelesen werden:

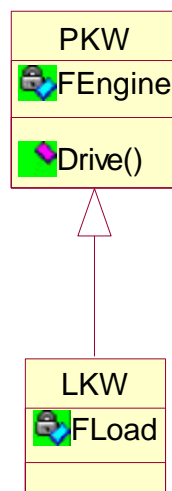
```
PROCEDURE SetLimit(NewLimit: Integer);  
  
BEGIN  
  
    IF PasswordOK THEN FLimit:=NewLimit;  
  
END;
```

Damit ist es nun nicht mehr möglich, daß unbefugte Personen das Kontolimit setzen. FNAME hingegen ist ein öffentlich zugängliches Attribut, d.h., daß dieses Attribut auch außerhalb der Klasse gelesen und geschrieben werden kann.

2.3. Vererbung

Eine der hilfreichsten Eigenschaften der objektorientierten Programmierung ist die Vererbung. Sie wird verwendet, um aus allgemeineren Klassen speziellere abzuleiten. Auf diese Weise können Prozeduren einer Basisklasse in den Nachfolgern ohne zusätzlichen Programmieraufwand verwendet werden.

Diesmal ein Beispiel mit Kraftfahrzeugen:



Die Basisklasse PKW mit dem Attribut FENGINE und der Prozedur DRIVE vererbt dem Nachfolger LKW alle Attribute und Prozeduren. Somit steht auch in der Klasse LKW die

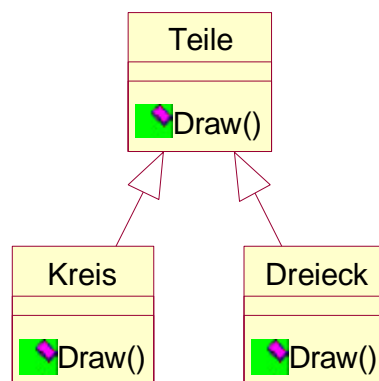
Prozedur DRIVE zur Verfügung, ohne neuerliche Definition. Zusätzlich kann dieser Nachfolger aber weitere Attribute und Prozeduren einbringen.

In der Grafik ist die Vererbung durch einen Generalisierungspfeil vom Nachfolger zum Vorgänger dargestellt. Wird ein Objekt LKW erzeugt, so werden zwar die Attribute und Prozeduren von PKW erzeugt, PKW selbst tritt aber in diesem Fall nicht als eigenständige Klasse auf. LKW ist also ein eigenes Objekt für sich.

Diese und die folgenden Grafiken entsprechen der UML (UNIFIED MODELING LANGUAGE) – Notation. UML ermöglicht die einheitliche Darstellung von Klassen und deren Beziehungen zu einander.

2.4. Polymorphismus

Die Grafik zeigt eine Klasse TEILE mit der virtuellen Prozedur DRAW. Virtuell heißt, daß diese Klasse das Interface definiert, aber nicht die Implementierung der Prozedur. KREIS und DREIECK erben von diesem Vorgänger, haben aber selbst auch DRAW – Prozeduren, die nun die eigentliche Implementierung dieser Prozedur enthalten.



Man kann nun einer Variablen vom Typ TEILE ein bestimmtes Objekt zuweisen und dieses zeichnen:

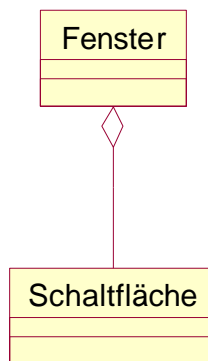
```
VAR
    Teil: Teile;
BEGIN
    Teil:=Kreis;
    Teil.Draw;
[...]
```

```
Teil:=Dreieck;  
Teil.Draw;  
END;
```

Im ersten Fall wird ein Kreis und im zweiten Fall ein Dreieck gezeichnet, obwohl in beiden Fällen die selbe Funktion DRAW verwendet wird. Dieses unterschiedliche Verhalten einer Prozedur abhängig vom Kontext nennt man Polymorphismus.

2.5. Aggregation

Es gibt jedoch auch Objekte, die für sich alleine nicht existieren können. Z.B. hat eine Schaltfläche auf einem Fenster nur dann Sinn, wenn das Fenster existiert.



Diese Beziehung zwischen Klassen wird Aggregation genannt. Das Fenster besitzt gewissermaßen die Schaltfläche oder anders ausgedrückt, die Schaltfläche ist Teil des Fensters und hat alleine keine Existenz. Wird das Fenster gelöscht, verschwindet auch die Schaltfläche.

In der Grafik ist dies durch eine Raute gekennzeichnet, die zur Hauptklasse zeigt.

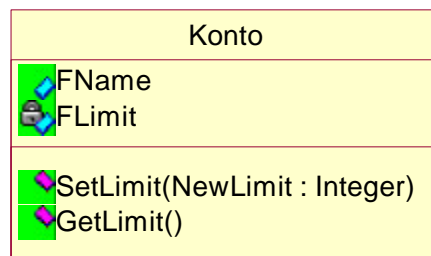
3. OOP in DELPHI

3.1. Einleitung

In diesem Kapitel werden die DELPHI - spezifischen objektorientierten Begriffe näher erläutert.

3.2. PROPERTIES

Unter PROPERTIES versteht man in DELPHI jene Attribute eines Objekts, die mit dem Schlüsselwort PROPERTY definiert wurden.



Für das obige Beispiel lautet die Klassendefinition:

TYPE

Konto = Class

PRIVATE

FLimit: Integer;

PUBLIC

FName: String;

PROCEDURE SetLimit(NewLimit: Integer);

PROCEDURE GetLimit();

PROPERTY FLimit: Integer READ GetLimit

WRITE SetLimit;

PROTECTED

PUBLISHED

END;

Das Objekt KONTO besitzt das PROPERTY FLIMIT. In DELPHI gibt es vier Zugriffsarten:

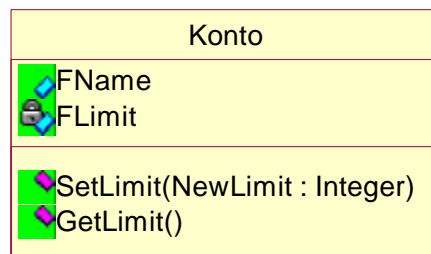
- PRIVATE
- PUBLIC
- PROTECTED
- PUBLISHED

FLIMIT steht im PRIVATE Bereich und ist daher außerhalb der Klasse nicht sichtbar. Im Gegensatz dazu steht FNAME im PUBLIC Bereich und ist auch außerhalb zu sehen. Der PROTECTED Bereich ist für Benutzer der Klasse nicht sichtbar, wohl aber für Klassen, die von dieser Klasse erben. Der letzte Bereich PUBLISHED ist ähnlich dem PUBLIC Bereich auch außen sichtbar, zusätzlich sind aber PROPERTIES in diesem Bereich im OBJECT INSPECTOR sichtbar. Der OBJECT INSPECTOR ist ein Fenster in DELPHIS Entwicklungsumgebung, mit dem man PROPERTIES von Objekten Werte zuweisen kann.

Der Unterschied zwischen dem direkt zugänglichen Attribut FNAME und dem PROPERTY FLIMIT besteht darin, daß FLIMIT sogenannte Triggermethoden GETLIMIT und SETLIMIT besitzt. Diese Methoden werden beim Lesen und Schreiben des Attributs automatisch ausgelöst und können daher beim Zugriff auf das Attribut weitere Aktionen auslösen. Z.B. kann die Triggermethode SETLIMIT eine Bereichsüberprüfung durchführen.

3.3.METHODEN

In DELPHI werden die Prozeduren und Funktionen einer Klasse METHODEN genannt.

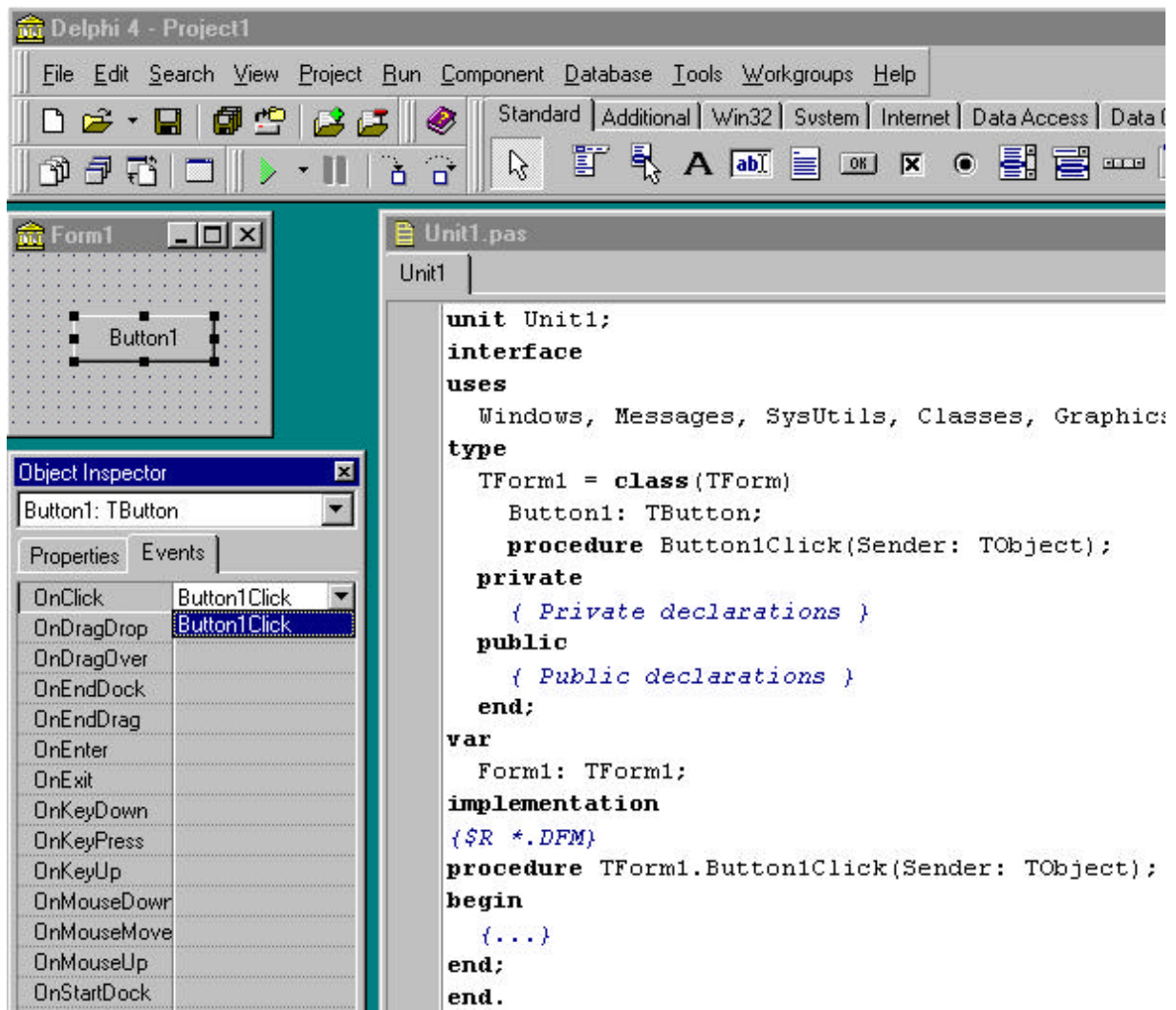


Im obigen Beispiel ist SETLIMIT und GETLIMIT eine METHODE der Klasse KONTO. Der Zugriff auf diese Methoden wird, wie bei den PROPERTIES, über die Bereiche PRIVAT, PUBLIC, PROTECTED und PUBLISHED geregelt.

3.4. EVENTS

Ereignisse, wie z.B. das Betätigen einer Schaltfläche oder der Ablauf eines TIMERS lösen in DELPHI sogenannte EVENTS aus. Auf diese Ereignisse kann in weiterer Folge reagiert werden.

Das Beispiel zeigt den CLICK EVENT, der durch Betätigen einer Schaltfläche ausgelöst wird. In der Prozedur BUTTON1CLICK wird dann auf dieses Ereignis reagiert:



The screenshot shows the Delphi 4 IDE interface. On the left, a form titled 'Form1' contains a button labeled 'Button1'. Below the form is the 'Object Inspector' window, which shows the 'Events' tab for 'Button1: TButton'. The 'OnClick' event is assigned to 'Button1Click'. On the right, the 'Unit1.pas' code editor shows the following Pascal code:

```

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics;
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
begin
  (...);
end;
end.

```

4. ORI im Entwicklungssystem DELPHI

4.1. Einleitung

Als Entwicklungssystem für die Implementierung der ORI – Architektur wurde DELPHI gewählt. DELPHI bietet in Bezug auf Objektorientierung, Umfang und Weiterentwicklung die beste Wahl. DELPHI bietet Unterstützung für COM/DCOM, abstrakte Datentypen, einen schnellen 32-Bit Compiler, METHOD OVERLOADING, dynamische Arrays und 64-Bit INTEGER. Das Kapitel beschäftigt sich mit dem INTEGRATED DEVELOPMENT ENVIRONMENT (IDE) und wie Anwendungen mit den ORI – Komponenten geschrieben werden können.

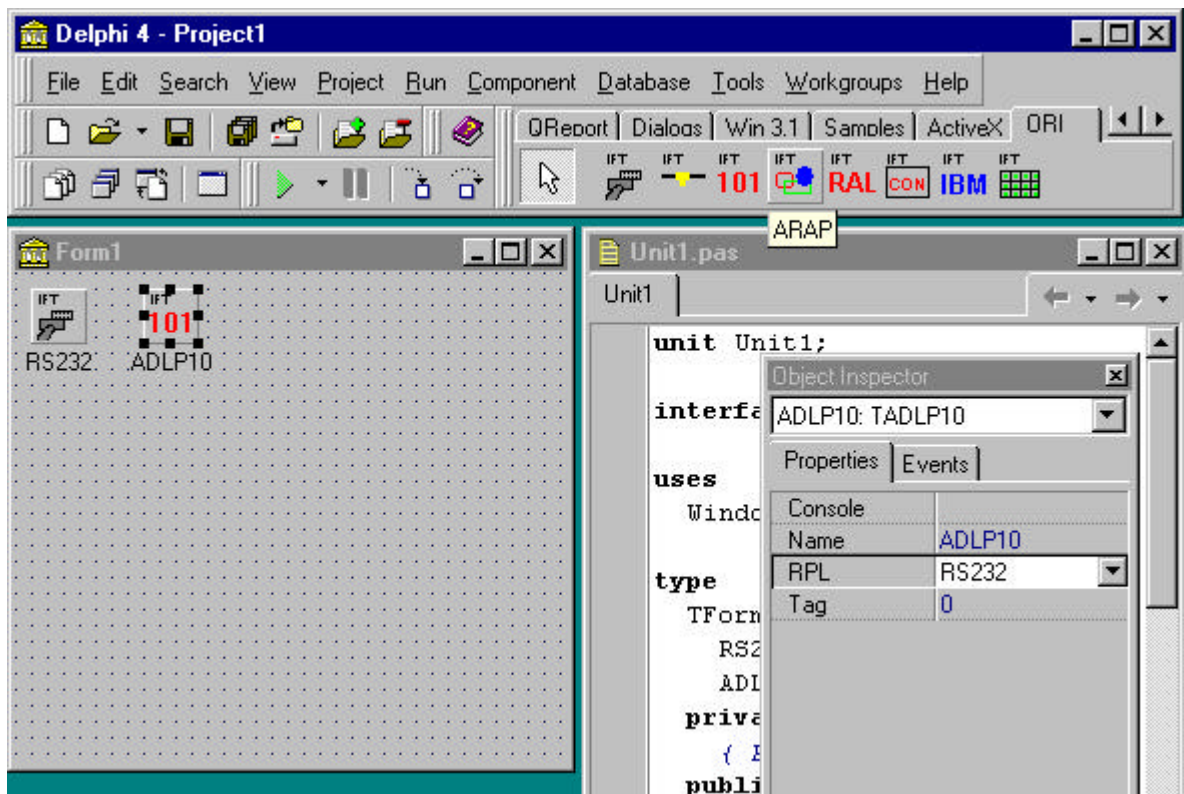
4.2. OPEN ROBOT INTERFACE (ORI) - ARCHITECTURE

Die ORI – Architektur ist ein Komponenten – orientiertes Schichtensystem zur Steuerung von Robotern. Es bietet in jeder dieser Schichten ein klar definiertes Interface. Auch die oberste Schicht bietet dieses Interface und stellt es Anwendungen zur Verfügung. ORI selbst ist also keine Anwendung, sondern besteht aus Komponenten, die das Programmieren von Anwendungen zur Steuerung von Robotern extrem vereinfacht. Details zum Aufbau von ORI finden sich im folgenden Kapitel „Schichtentrennung und ORI – Komponenten“.

Natürlich ist diese Architektur nicht auf Roboter beschränkt, sondern kann auch für andere Maschinen verwendet werden.

4.3. DELPHIS IDE

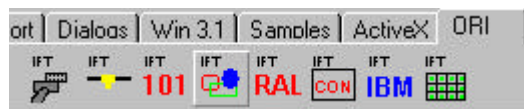
IDE steht für INTEGRATED DEVELOPMENT ENVIRONMENT, also DELPHIS Entwicklungsumgebung. Diese Umgebung ermöglicht das Erstellen von Oberflächen (FORMS) , das Programmieren von Code in einem Editor, Editieren der Eigenschaften von Objekten im OBJECT INSPECTOR, Übersetzen des Codes sowie einen hervorragenden Debugger:



Auf der Form sind bereits zwei Komponenten aus dem ORI - PACKAGE zu sehen. Diese sind aus DELPHIS COMPONENT LIBRARY zu wählen.

4.4. Komponenten in der IDE

Die Komponenten im Detail:



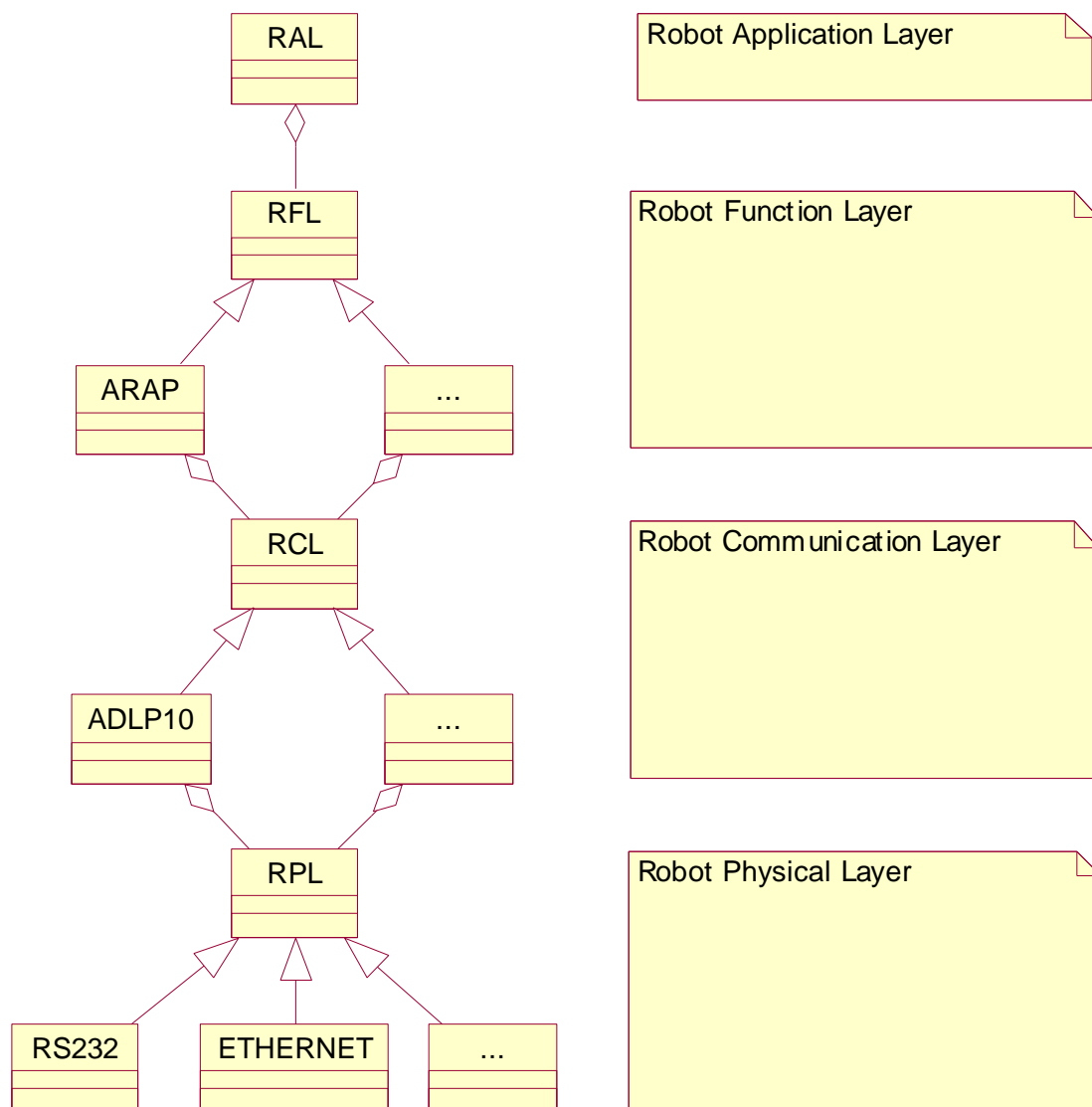
Von links nach rechts stehen RS232 -, ETHERNET -, ADLP10 -, ARAP - und die RAL - Komponente zu Verfügung.

Für Textausgaben aus diesen Komponenten ist eine CONSOLE - Komponente vorhanden. Auch weitere Roboter können durch Komponenten eingebunden werden (IBMROBOTER). Zuletzt kann die Funktionalität auch durch andere Komponenten, wie z.B. Palettenfunktionen, erweitert werden (ROBOTERPALETTE). Diese letzteren, Roboter - unabhängigen Komponenten stellen bereits Erweiterungen des ORI - Konzepts dar und werden in dieser Arbeit nicht behandelt.

5. Schichtentrennung und ORI - Komponenten

5.1. Einleitung

Dieses Kapitel beschreibt die logische Aufteilung in Schichten sowie die objektorientierte Aufteilung in Klassen:



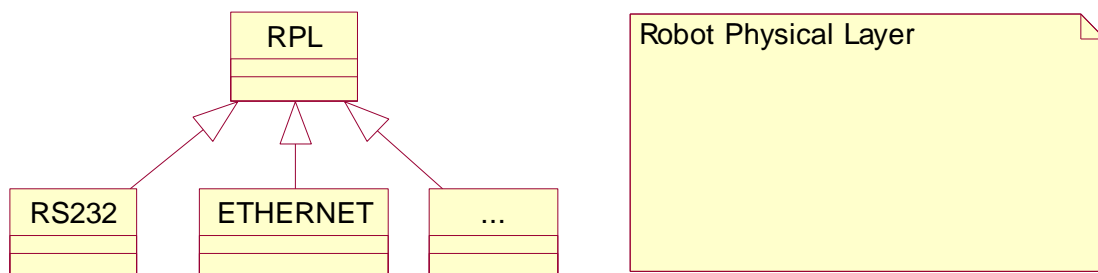
Logisch und funktionell gliedert sich der Aufbau in drei Schichten. Die unterste Schicht ist der ROBOT PHYSICAL LAYER, die RPL - Schicht. Sie ist für das Übertragen von Bits zuständig. Zur Wahl steht die RS232 - Komponente bzw. in Zukunft auch eine ETHERNET Komponente. Der darüberliegende ROBOT COMMUNICATION LAYER, die RCL - Schicht, verwendet die RPL - Schicht und ist selbst für das Übertragen von

Telegrammen zuständig. Der für die Roboterfunktionalität zuständige, darüberliegende ROBOT FUNCTION LAYER, die RFL - Schicht, bedient sich der RCL - Schicht und stellt Roboterfunktionen z.B. für die Bewegung zur Verfügung. Der ganz oben liegende ROBOT APPLICATION LAYER, die RAL – Schicht, dient als einheitliche Schnittstelle zu den Anwendungen und hat selbst keine interne Funktionalität.

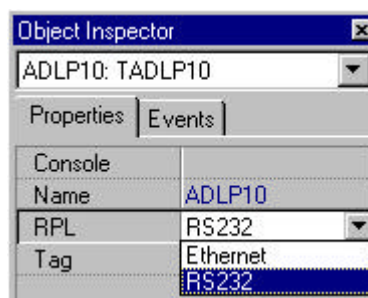
RPL, RCL und RFL sind abstrakte Klassen. D.h., daß diese Klassen nur das Interface definieren, aber nicht die Implementierung. So wird z.B. in der Klasse RFL das Interface der Methode MOVE definiert, die tatsächliche Implementierung erfolgt aber dann in der ARAP - Klasse.

5.2. ROBOT PHYSICAL LAYER (RPL)

RPL vererbt das abstrakte Interface entweder an die RS232 - Komponente oder in Zukunft z.B. auch an die Ethernet - Komponente, welche dieses dann implementieren.

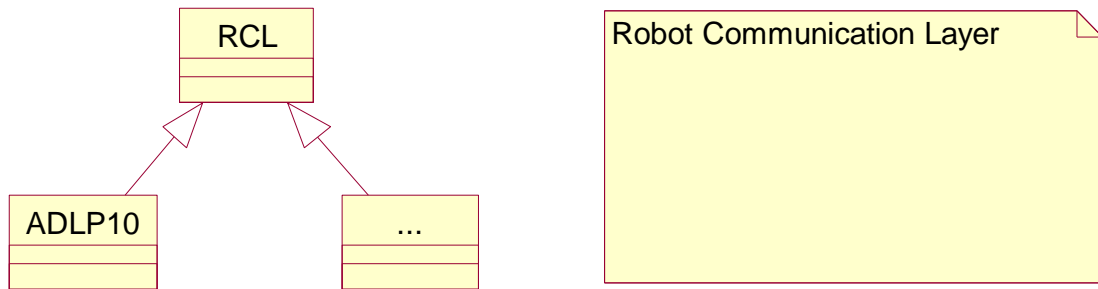


In der darüberliegenden Schicht (ADLP10 - Protokoll) kann die aktuelle Komponente durch ein PROPERTY im OBJECT INSPECTOR festgelegt werden (AGGREGATION):

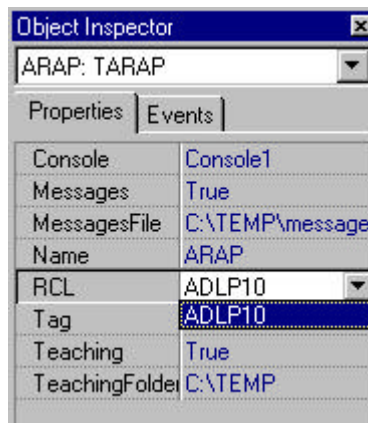


5.3. ROBOT COMMUNICATION LAYER (RCL)

RCL vererbt an das eingestellte Kommunikationsprotokoll, hier ist ADLP10 ausgewählt.



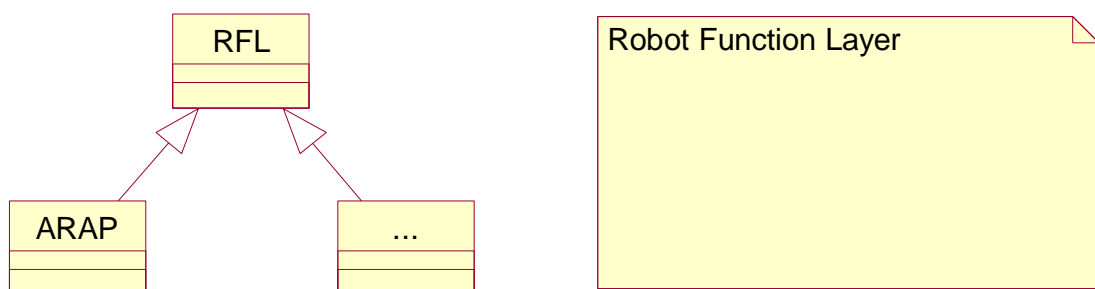
Wieder kann diese Einstellung über den OBJECT INSPECTOR gewählt werden:



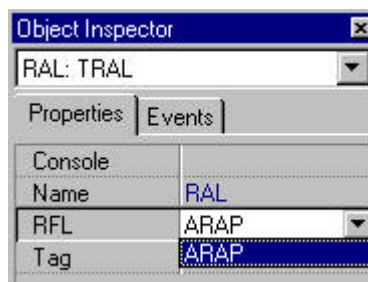
Die ARAP Komponente (darüberliegende Schicht) besitzt dazu das PROPERTY RCL.

5.4. ROBOT FUNCTION LAYER (RFL)

Als letzte Schicht mit Funktionalität vererbt RFL an das gewählte ARAP Protokoll:



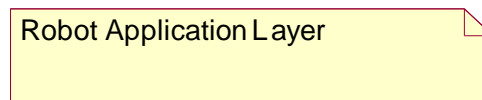
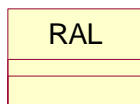
Im darüberliegenden RAL kann das gewünschte Protokoll aggregiert werden:



Wieder ist dazu das PROPERTY RFL in der RAL - Komponente vorhanden.

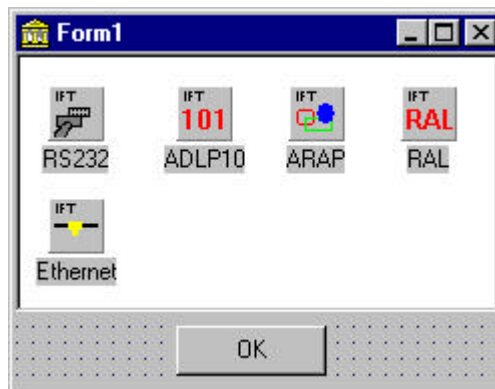
5.5. ROBOT APPLICATION LAYER (RAL)

RAL stellt die Schnittstelle zu den eigentlichen Anwendungen dar (API) und macht Anwendungen dadurch portabel.



5.6. ORI - Komponenten

Eine Applikation benötigt also (mindestens) eine Komponente aus jeder Schicht auf der FORM. Hier ein Beispiel:



Die Komponente ETHERNET ist hier nicht notwendig, wenn man nur über die serielle Schnittstelle kommunizieren will.

6. Roboter Hardware

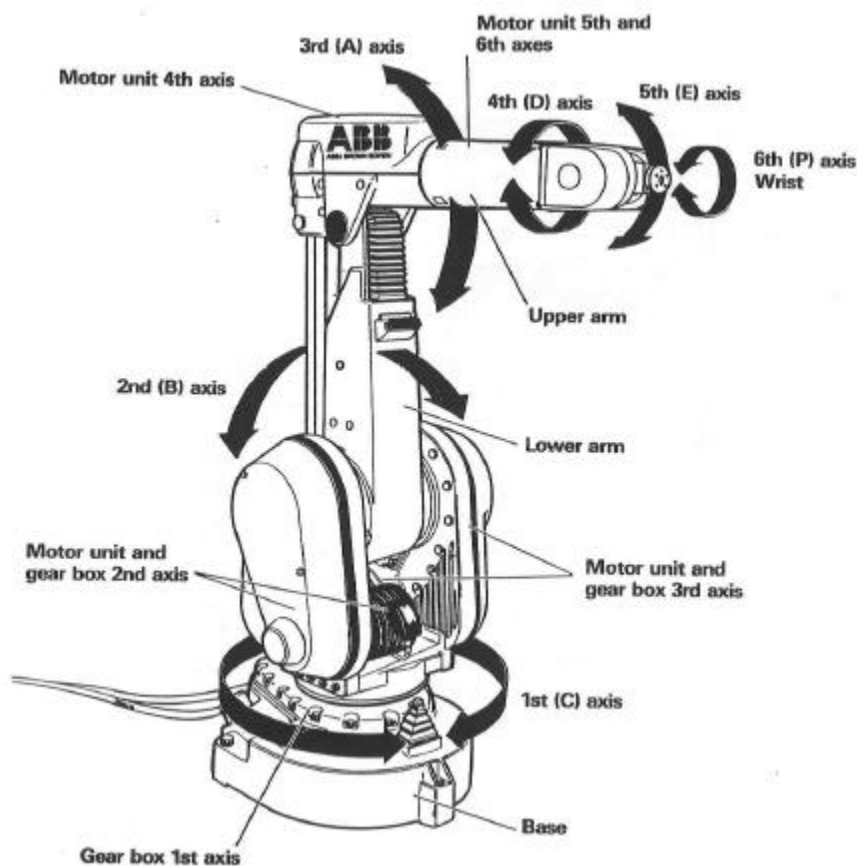
6.1. Einleitung

Dieses Kapitel widmet sich dem Roboter. Es erklärt den mechanischen Teil, die Steuerungseinheit und die Schnittstelle für den Computer.

6.2. Mechanischer Teil

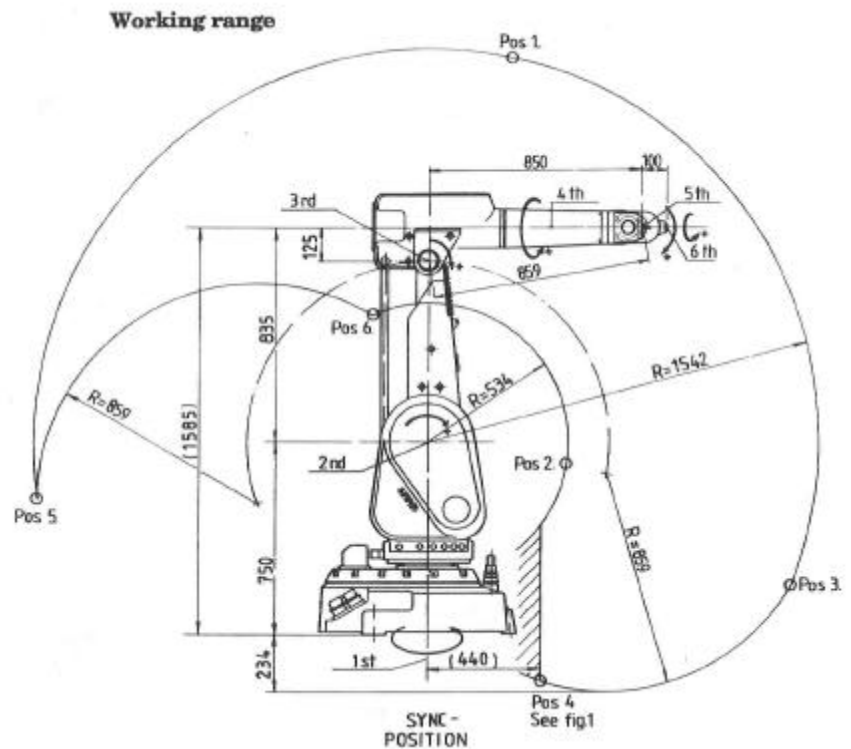
Der ABB IRB 2000 ist ein 6 Achsen - Roboter. Er wird hauptsächlich zum Schweißen, Kleben, Palettieren, Laser - oder Wasserstrahl - Schneiden und zum Zusammensetzen von Bauteilen verwendet.

Die erste Achse ermöglicht eine Drehung des gesamten Roboters um die eigene, vertikale Achse. Die zweite Achse betätigt den unteren Arm, die dritte Achse den oberen Arm. Die vierte Achse ermöglicht ein Drehen des oberen Arms. Die fünfte Achse kippt das Handgelenk und die sechste Achse dreht das Handgelenk:



6.3. Arbeitsbereich

Die Achsen können nur innerhalb eines bestimmten Bereichs verfahren werden. Wird der Roboter außerhalb dieses sogenannten Arbeitsbereichs verfahren, stoppt die Steuerung die Bewegung. Am Ende der Achsenbewegung liegt zusätzlich ein mechanischer Anschlag.

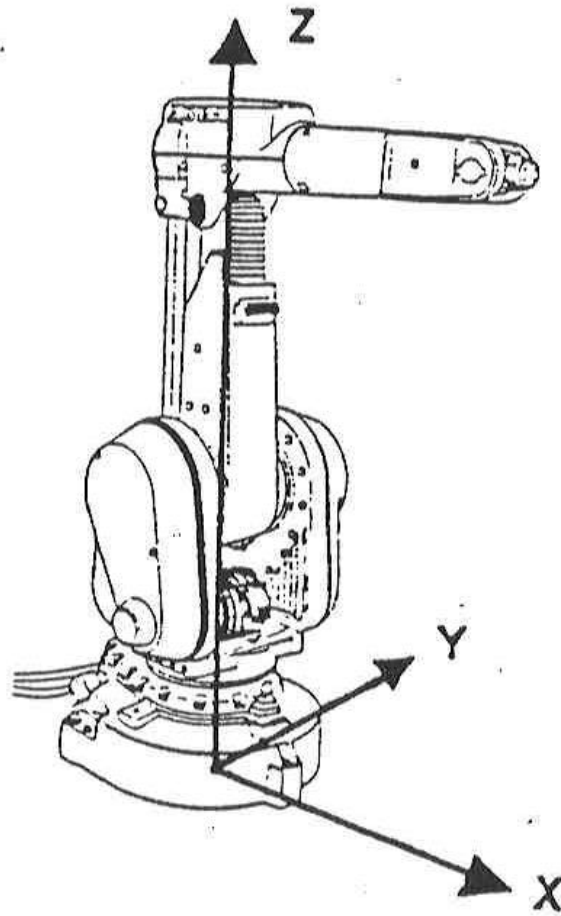


Die Maßzahlen verstehen sich in [mm].

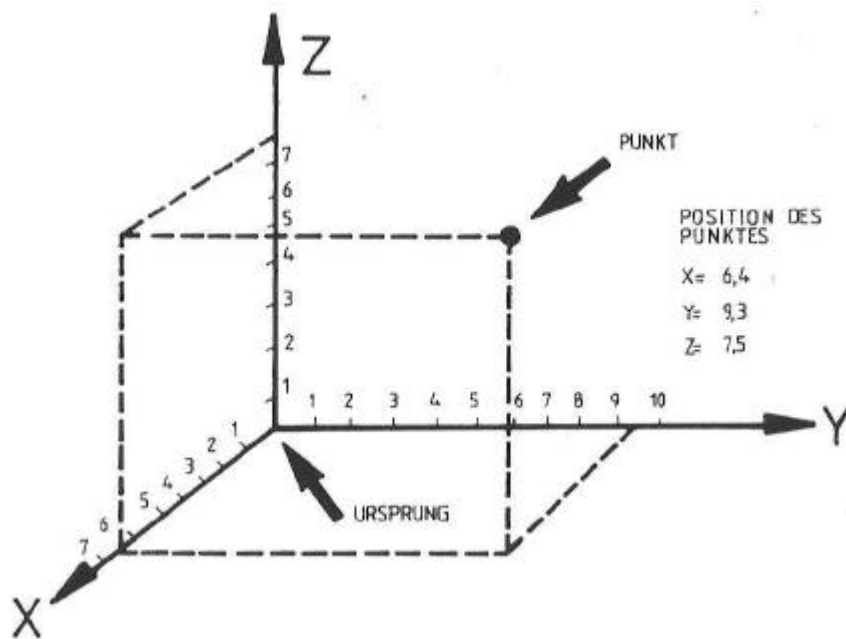
6.4. Koordinatensysteme

6.4.1. Rechtwinkeliges sockelorientiertes Koordinatensystem

Dieses Koordinatensystem wird auch als Kartesisches Koordinatensystem bezeichnet. Der Ursprung liegt im Robotersockel:



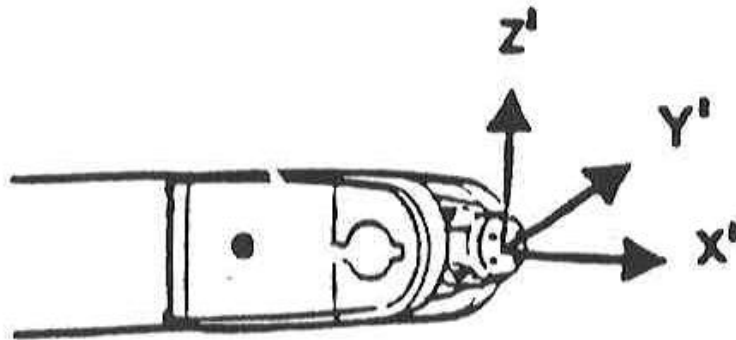
Das Bild zeigt die Synchronisationsposition des Roboters – das ist die Ausgangsstellung.



Koordinatenangaben betreffen den sogenannten TOOL CENTER POINT (TCP), das ist jener Punkt eines Werkzeugs, der für die Position des Werkzeugs entscheidend ist. Der TCP muß für jedes Werkzeug in der Steuerung definiert werden.

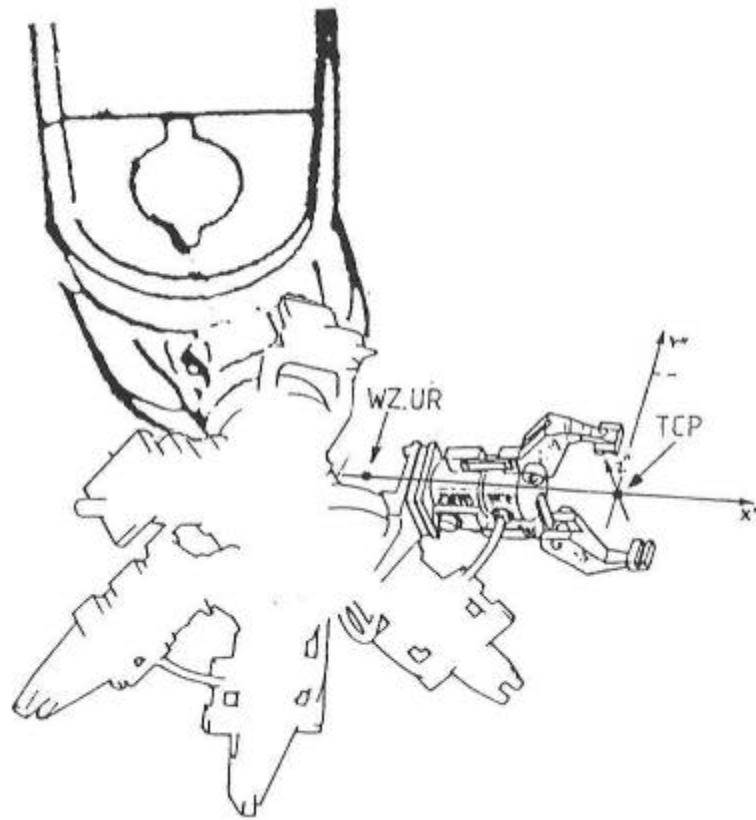
6.4.2. Rechtwinkeliges handgelenkorientiertes Koordinatensystem

Dieses Koordinatensystem bezieht sich auf das Handgelenk des Roboters.



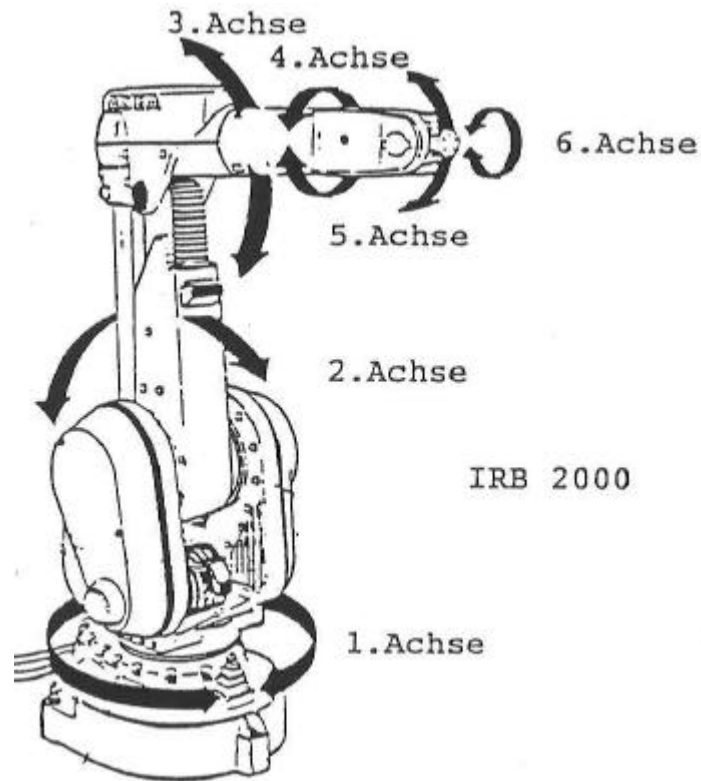
6.4.3. Rechtwinkeliges werkzeugorientiertes Koordinatensystem

Dieses Koordinatensystem entsteht nach der Definition eines Werkzeugursprunges (WZ.UR) zu einem TCP aus dem handgelenkorientierten Koordinatensystem. Der zum Werkzeug gehörende TCP ist der Ursprung des Koordinatensystems.



6.4.4. Roboterachsenkoordinatensystem

Zuletzt gibt es auch noch das Roboterachsen - Koordinatensystem. Hier bildet jede Roboterachse eine Achse des Koordinatensystems:

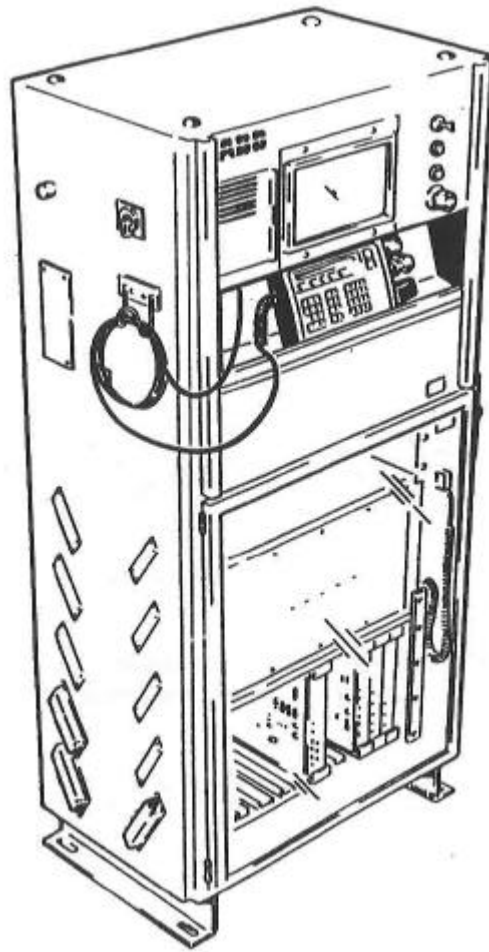


6.5. Steuerung

Die Steuerung ist ein wesentlicher Bestandteil des „Roboters“. Erst durch sie wird das Bewegen des mechanischen Teils ermöglicht. In der Steuerung können Programme ablaufen oder man bedient den Roboter händisch über die PROGRAMMING UNIT.

Die genaue Bezeichnung der verwendeten Steuerung lautet ABB ROBOT CONTROL SYSTEM S3. In die Steuerung können verschiedene Einschübe installiert werden, z.B. ist ein Einschub für die RS232 - Schnittstelle vorhanden.

Die Steuerung verfügt auch über ein VISION SYSTEM, mit dem über eine Kamera das Erkennen und Hantieren von Bauteilen auf optischer Basis ermöglicht wird.



Im Bild ist auch gut die PROGRAMMING UNIT zu sehen, die im folgenden Kapitel erläutert wird.

6.6. PROGRAMMING UNIT

Die PROGRAMMING UNIT ist der Bedienteil der Steuerung. Sie kann von der Steuerung abgenommen werden, da sie mit einem langen Kabel angeschlossen ist.



Mit dieser Konsole kann nicht nur der Roboter bewegt werden, sondern es können alle Funktionen des Roboters gesteuert werden. Man kann auch Programme über diese Einheit eingeben, diese auf Diskette abspeichern und ablaufen lassen. Natürlich ist diese Art der Programmierung sehr mühsam und mit einer Programmierung am Computer nicht vergleichbar.

6.7. Schnittstellen

Die Steuerung S3 verfügt in der vorhandenen Ausführung über eine RS232C - Schnittstelle, die auch als V24 bzw. V28 konfiguriert werden kann.

Das Kabel einer RS232 - Verbindung besteht aus 9 Leitungen und einer Schirmung:

Abkürzung	Erklärung	Nr. bei 9 Leitungen	(Nr. bei 25 Leitungen)
CD	Verbindung	1	8
RX	Daten empfangen	2	3
TX	Daten senden	3	2
DTR	Fertigmeldung Terminal	4	20
S.GND	Signalerde	5	7
DSR	Fertigmeldung Daten	6	6
RTS	Sendeaufforderung	7	4
CTS	Sendefreigabe	8	5
RI	Signal	9	22
GND	Erde	Schirmung	1

Wie diese Schnittstelle mit dem Computer verbunden werden muß, ist im folgenden Kapitel „RS232“ erläutert.

7. Roboter Protokolle

7.1. Einleitung

Dieses Kapitel beschäftigt sich mit dem prinzipiellen Aufbau der Schichten und deren Schnittstellen. Erst im nächsten Kapitel wird die tatsächliche Implementierung beschrieben.

Die Kommunikation zwischen Roboter und Computer ist in drei Schichten unterteilt. Die unterste Schicht ist die physikalische Schicht (PHYSICAL LAYER). Darüber liegt die Kommunikationsschicht (COMMUNICATION LAYER) und die oberste Schicht ist die funktionelle Schicht (FUNCTION LAYER). Jede Schicht hat eine spezielle Aufgabe und eine genau definierte Schnittstelle nach oben und unten:

Anwendung	Roboter
Funktionelle Schicht (3)	Funktionelle Schicht (3)
Kommunikationsschicht (2)	Kommunikationsschicht (2)
Physikalische Schicht (1)	Physikalische Schicht (1)
Übertragungsmedium	Übertragungsmedium

7.2. RS232

RS232 ist ein Protokoll für die physikalische Schicht. Dieses Protokoll dient zur seriellen Übertragung von Bits zwischen zwei Kommunikationspartnern über ein Kabel.

7.2.1. Hardwareverbindung

Das Kabel einer RS232 - Verbindung besteht aus 9 Leitungen und einer Schirmung:

Abkürzung	Erklärung	Nr. bei 9 Leitungen	(Nr. bei 25 Leitungen)
CD	Verbindung	1	8
RX	Daten empfangen	2	3
TX	Daten senden	3	2

DTR	Fertigmeldung Terminal	4	20
S.GND	Signalerde	5	7
DSR	Fertigmeldung Daten	6	6
RTS	Sendeaufforderung	7	4
CTS	Sendefreigabe	8	5
RI	Signal	9	22
GND	Erde	Schirmung	1

Die einzelnen Leitungen müssen folgendermaßen verbunden werden:

Roboter	Roboter	Kabel	Computer
CD	DSR, DTR		
RX			TX
TX			RX
DTR	CD, DSR		
S.GND			S.GND
DSR	CD, DTR		
RTS	CTS		
CTS	RTS		

Will man den Datentransfer auf der Verbindung beobachten, benötigt man ein Y-Kabel. Die SNIFFER – Seite wird mit der zu beobachtenden Leitung parallel angeschlossen, die beiden anderen Enden werden in die seriellen Ports eines Beobachtungsrechners geleitet.

Als Beobachtungssoftware kann z.B. EASYCOM von BRIAN THORNE verwendet werden. Das Kabel ist folgendermaßen zu belegen:

Sniffer	Kabeln	COM1	COM2
CD		RI	
RX			RX
TX		RX	
DTR		DSR	
S.GND		S.GND	S.GND
DSR			DSR
RTS		CTS	
CTS			CTS
RI			RI

7.2.2. Konfiguration

Für die Kommunikation zwischen ABB - Roboter und Computer, genauer zwischen RS232 - Schnittstelle der Steuerung des Roboters und der RS232 - Schnittstelle des Computers, werden folgende Einstellungen verwendet:

- 9600 Bit/s

Das ist die Übertragungsgeschwindigkeit.

- 8 Bit Daten

Zu diesen 8 Bit Daten, das entspricht genau einem ASCII - Zeichen, kommt ein Paritäts - und ein Stop - Bit hinzu:

- Gerade Parität
- 1 Stop Bit

Es wird keine Hardware - oder Software - Flußkontrolle verwendet:

- Keine Flußkontrolle

7.3.ADLP10

7.3.1. Einleitung

Das ABB DATA LINK PROTOCOL (ADLP) ist ein serielles, asynchrones Kommunikationsprotokoll zwischen zwei Kommunikationspartnern. Es dient zur gesicherten Übertragung von Daten (Zeichen). Die sendende Station wird MASTER genannt, die empfangende Station SLAVE. Sowohl der Computer wie auch die Steuerung des Roboters kann MASTER sein.

Die Kommunikation findet HALF DUPLEX statt, d.h., daß immer nur eine Station zu einem Zeitpunkt senden kann.

Die Information, die übertragen werden soll, ist meist durch das darüberliegende Protokoll in sogenannte Telegramme unterteilt. Damit wird in einem Kommunikationsvorgang die Information in mehreren Telegrammen übertragen.

7.3.2. Steuerzeichen

Im ADLP10 - Protokoll werden folgende Steuerzeichen verwendet:

ENQ – ENQUIRY:

Mit diesem Zeichen beginnt die sendende Station, d.h. der MASTER, die Kommunikation. Die empfangende Station, d.h. der SLAVE, muß mit ACK reagieren, um die Kommunikation aufzubauen.

ACK – ACKNOWLEDGE:

Dies ist das Zeichen für positive Bestätigung des SLAVES.

WACK – WAIT AND ACKNOWLEDGE:

Dieses Zeichen bestätigt ebenfalls positiv, aber signalisiert gleichzeitig, daß die empfangende Station keine weiteren Daten derzeit aufnehmen kann.

RVI – REVERSE INTERRUPT:

Auch dieses Zeichen bestätigt positiv. Gleichzeitig signalisiert dieses Zeichen aber auch dem MASTER, daß der SLAVE selbst Daten zu senden hat und daher MASTER werden möchte.

NAK – NEGATIV ACKNOWLEDGE:

Der SLAVE signalisiert mit diesem Zeichen eine negative Bestätigung. Das ist z.B. keine Empfangsbereitschaft des SLAVES oder eine fehlerhafte Datenübertragung.

DLE – DATA LINK ESCAPE:

Durch dieses Zeichen signalisiert der MASTER dem SLAVE, daß das nächste Zeichen als Steuerzeichen und nicht als Daten zu interpretieren ist.

STX (even/odd) – START OF TEXT:

Damit wird der Beginn des Textes, d.h. der Daten signalisiert. STX gibt es in gerader und ungerader Ausführung, um sequentielles Markieren zu ermöglichen.

ETX – END OF TEXT:

Dies signalisiert das Ende des Textes, eben das Ende der Daten.

EOT – END OF TRANSMISSION:

Mit diesem Zeichen wird die Kommunikation abgeschlossen.

Die Kodierung obiger Zeichen ist in der ASCII - Code Tabelle im Anhang nachzulesen.

7.3.3. Kommunikationsphasen

Die Kommunikation teilt sich in 3 Phasen:

7.3.3.1. Kommunikationsaufbau

In der ersten Phase wird die Kommunikation aufgebaut. Der MASTER sendet zum SLAVE das Zeichen ENQ. Wenn der SLAVE empfangsbereit ist, sendet er ACK zurück. Damit ist die erste Phase ordnungsgemäß abgeschlossen.

Die Antwort des SLAVE kann jedoch auch WACK oder RVI sein. Bei WACK kann der SLAVE derzeit nicht empfangen – z.B. weil sein Empfangspuffer voll ist. Bei RVI wünscht der SLAVE selbst MASTER zu werden, da er Daten senden möchte.

Senden sowohl Computer wie auch Roboter gleichzeitig ENQ, so hat der Computer Vorrang vor dem Roboter.

7.3.3.2. Datentransfer

Jedes Telegramm, das der MASTER übertragen soll, wird in die Zeichen DLE - STX und DLE - ETX eingeschlossen.

Ein in den Daten vorkommendes Zeichen DLE wird verdoppelt, um es von einem Steuerzeichen unterscheiden zu können. D.h., wird die Zeichenfolge XXX – DLE - YYY empfangen, wobei XXX und YYY nicht DLE ist, so ist das Zeichen DLE als DATA LINK ESCAPE zu interpretieren. Wird DLE - DLE empfangen, so ist ein DLE zu entfernen und das Verbleibende als Daten zu interpretieren.

Nach jedem übertragenen Telegramm wird nach den Zeichen DLE – ETX auch noch eine Blockchecksumme gesendet.

Der SLAVE muß darauf mit ACK für positive Bestätigung reagieren. Danach folgt das nächste Telegramm oder der Kommunikationsabbau. Andere Antworten sind WACK, RVI oder NACK. Bei WACK wurde das zuletzt empfangene Telegramm positiv bestätigt, aber die Bereitschaft zum Empfang aufgehoben. Der MASTER kann eine gewisse Zeit warten und neuerlich versuchen dem SLAVE Daten zu schicken oder er beginnt mit dem Kommunikationsabbau. Bei RVI bekommt der MASTER ebenfalls eine positive Bestätigung, wird aber gleichzeitig um Abgabe der MASTER - Funktion gebeten. Daher folgt meist der Kommunikationsabbau. NACK ist die negative Bestätigung, es wird das zuletzt gesendete Telegramm wiederholt.

7.3.3.3. Kommunikationsabbau

Den Abbau der Kommunikation bestimmt der MASTER durch das Steuerzeichen EOT. Danach muß die Kommunikation neu aufgebaut werden.

7.3.4. Sicherungsmechanismen

Die Kommunikation wird auf mehrere Arten gesichert:

7.3.4.1. Horizontale Parität

Jedes einzelne zu übertragende Zeichen wird durch ein Paritätsbit gesichert. Das Zeichen selbst besteht aus 8 Bit Daten. Daraus wird mittels gerader Parität das Paritätsbit mittels Modulo-2 Summe errechnet.

Dieser Sicherungsmechanismus ist bereits im Betriebssystem bzw. der RS232 - Implementierung vorhanden und muß daher nur richtig konfiguriert werden:

- 8 Bit Daten
- Gerade Parität
- 1 Stop Bit

7.3.4.2. Vertikale Parität (Blockchecksumme)

Die vertikale Parität oder Blockchecksumme (BCS) wird für jeden Block berechnet. Ein Block entspricht einem Telegramm. Genau beginnt ein Block nach den Zeichen DLE – STX exklusive und endet mit den Zeichen DLE - ETX inklusive. Das Zeichen DLE vor ETX wird dabei nicht berücksichtigt. Ebenso ein in den Daten vorkommendes DLE – DLE wird nur einmal berücksichtigt.

Die Blockchecksumme wird durch vertikale, bitweise Modulo-2 Summe der Zeichen errechnet und gleich anschließend an die Zeichen DLE – ETX eines Telegramms gesendet.

7.3.4.3. Sequentielles Markieren

Durch sequentielles Markieren wird es dem SLAVE ermöglicht, ein Folge - Telegramm von einer Wiederholung zu unterscheiden. Das erste Telegramm in einer Kommunikation erhält am Beginn das Zeichen STX (even), das nächste STX (odd) und danach abwechselnd. Die Even/Odd Markierung erfolgt über das MSB von STX:

Befehl	Binär	Dezimal
STX (even)	00000010	2
STX (odd)	10000010	130

7.3.5. Kommunikations-Beispiele

Die folgenden Tabellen sind so zu interpretieren, daß z.B. ein in der Spalte MASTER eingetragenes Zeichen ENQ als „MASTER sendet ENQ an den SLAVE“ zu lesen ist.

7.3.5.1. Kommunikation in einem Telegramm

Die zu übertragenden Daten können in einem Telegramm übertragen werden:

MASTER		SLAVE
ENQ		
		ACK
DLE		
STX (even)		
„Daten“		
DLE		
ETX		
BCS		

		ACK
EOT		

7.3.5.2. Kommunikation in mehreren Telegrammen

Die zu übertragenden Daten werden in mehreren Telegrammen übertragen:

MASTER		SLAVE
ENQ		
		ACK
DLE		
STX (even)		
„Daten“		
DLE		
ETX		
BCS		
		ACK
DLE		
STX (odd)		
„Daten“		
DLE		
ETX		
BCS		
		ACK
DLE		
STX (even)		
„Daten“		
DLE		

ETX		
BCS		
		ACK
EOT		

7.3.5.3. Übertragungsfehler

Bei der Übertragung der Daten tritt ein Fehler auf:

MASTER		SLAVE
ENQ		
		ACK
DLE		
STX (even)		
„Daten“		
DLE		
ETX		
BCS		
		NAK
DLE		
STX (even)		
„Daten“		
DLE		
ETX		
BCS		
		ACK
DLE		
STX (odd)		

„Daten“		
DLE		
BCS		
ETX		
		ACK
EOT		

7.3.5.4. TIMEOUT

Nach dem Senden der Daten erfolgt keine Rückmeldung innerhalb des TIMEOUTS.
(Beim ABB - Roboter ist der TIMEOUT auf 5 Sekunden gestellt.)

MASTER		SLAVE
ENQ		
		ACK
DLE		
STX (even)		
„Daten“		
DLE		
ETX		
BCS		
DLE		
STX (even)		
„Daten“		
DLE		
ETX		
BCS		

		ACK
DLE		
STX (odd)		
„Daten“		
DLE		
ETX		
BCS		
		ACK
EOT		

7.4. ARAP

7.4.1. Einleitung

Das ABB ROBOT APPLICATION PROTOCOL (ARAP) ist ein funktionelles Protokoll zum Steuern von Robotern. In der ABB - Notation wird es bereits als das „Anwendungsprotokoll“ verstanden. Es bedient sich der darunterliegenden Kommunikationsschicht, dem ADLP10 Protokoll. Die zu versendende Information kann von ARAP in mehrere Telegramme zerlegt werden, da ein Telegramm nach ARAP maximal 128 Bytes lang sein darf.

7.4.2. Telegrammaufbau

Jedes Telegramm besteht aus einem 8 Byte großen Kopf und einem maximal 120 Byte großen Datenfeld. Diese maximal 128 Byte werden von ADLP10 in DLE – STX und DLE – ETX – BCS eingebettet und versendet. Ein Telegramm ist also genau das, was durch das ADLP10 Protokoll in DLE – STX und DLE – ETX – BCS eingebettet wird.

Telegramm:

Bezeichnung	Größe
Kopf	Genau 8 Byte
Datenfeld	Maximal 120 Byte

7.4.2.1. Kopf

Der Kopf ist bei allen Telegrammen vorhanden und immer 8 Byte lang. Jedes Byte hat eine feste Bedeutung:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE								4
-	-	-	ORT	MLI	RS	T	T	5
FUNCTION SUFFIX								6
								7

NOB – NUMBER OF BYTES:

Dieses Feld gibt die Anzahl Bytes inklusive Kopf im aktuellen Telegramm an. Für diese Information sind zwei Byte vorgesehen.

DESTINATION ADDRESS:

Gibt die Adresse des Telegramm - Empfängers an. Die Adresse für den Roboter kann in der Robotersteuerung definiert werden. Als Standardwert sollte Eins für den Roboter und Null für den Computer verwendet werden.

SOURCE ADDRESS:

Dies ist die Adresse des Senders eines Telegramms. Die Adresse für den Roboter kann in der Robotersteuerung definiert werden. Als Standardwert sollte Eins für den Roboter und Null für den Computer verwendet werden.

FUNCTION CODE:

Spezifiziert die Funktion, die ausgeführt werden soll, durch eine Nummer.

ORT – ORIENTATION:

Bei auf Null gesetztem Bit werden handgelenkorientierte Koordinaten verwendet, bei Eins werkzeugorientierte Koordinaten.

MLI – MESSAGE LENGTH INDICATOR:

Dieses Bit ist Null, falls die gesamte Information in einem Telegramm enthalten ist oder dieses Telegramm das letzte in einer Folge von Telegrammen ist. MLI ist auf Eins gesetzt, falls die Information in mehreren, dem aktuellen Telegramm folgenden Telegrammen übertragen wird.

RS – RESPONSE STATUS:

Bei auf Null gesetztem Bit entspricht dies einer positiven Bestätigung eines Befehls, bei Eins einer negativen Bestätigung. Bei einer negativen Rückmeldung enthält das Datenfeld den ERRORCODE. Dies ist die Rückmeldung des ARAP - Protokolls und nicht zu verwechseln mit der Rückmeldung des ADLP10 - Protokolls!

TT – TELEGRAM TYPE:

Diese zwei Bits geben die Art des Telegramms an. Eins entspricht einem COMMAND, Zwei einem RESPONSE und Drei einer SPONTANEOUS MESSAGE.

FUNCTION SUFFIX:

Hier wird je nach Funktion näher spezifiziert, wie sie ausgeführt werden soll. Auch für diese Information sind zwei Byte vorgesehen.

7.4.2.2. Datenfeld

Im Normalfall enthält das Datenfeld die Daten mit maximal 120 Byte:

7	6	5	4	3	2	1	0	BIT / BYTE
DATA								8
DATA								9
DATA								10
...								...
DATA								127

Im Fehlerfall enthält es aber den Fehlercode und ist genau 2 Byte lang:

7	6	5	4	3	2	1	0	BIT / BYTE
ERRORCODE								8
ERRORCODE								9

7.4.3. Telegrammtypen

Es gibt drei verschiedene Arten von Telegrammen:

Telegramm	TT binär	TT dezimal
COMMAND	01	1
RESPONSE	10	2
SPONTANEOUS MESSAGE	11	3

FUNCTION SUFFIX	6
	7
PROGRAM NUMBER	8
	9

FUNCTION SUFFIX:

Null bedeutet START FROM BEGINNING und Eins bedeutet START FROM LAST STOP.

PROGRAM NUMBER:

Nummer des Programms, 0-9999. Nur relevant, wenn die FUNCTION SUFFIX Null ist.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=10								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=2								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER (ERRORCODE)								8
								9

RESPONSE ist die Rückmeldung des Roboters.

7.4.4.1. Daten

In den FUNCTION CODES werden folgende Datenformate verwendet:

Koordinaten

7	6	5	4	3	2	1	0	BIT / BYTE
X-Koordinate								0
								1
Y-Koordinate								2
								3
Z-Koordinate								4
								5
Q1-Koordinate								6
								7
Q2-Koordinate								8
								9
Q3-Koordinate								10
								11
Q4-Koordinate								12
								13
Koordinate der 6. Achse (TCP Nummer)								14
-----								15
								16
								17
Koordinate der 7. Achse								18

	19
	20
	21
Koordinate der 8. Achse	22
	23
	24
	25
Koordinate der 9. Achse	26
	27
	28
	29
Koordinate der 10. Achse	30
	31
	32
	33
Koordinate der 11. Achse	34
	35
	36
	37
Koordinate der 12. Achse	38
	39
	40
	41

TCP Register

7	6	5	4	3	2	1	0	BIT / BYTE
X-Koordinate								0
								1
Y-Koordinate								2
								3
Z-Koordinate								4
								5
X1-Koordinate								6
								7
Y1-Koordinate								8
								9
Z1-Koordinate								10
								11

RESPONSE ERRORCODES

ERRORCODE	Erklärung
0	Nicht spezifiziert
1	Programm läuft
2	Falsche FUNCTION SUFFIX
3	Falsche PROGRAM NUMBER
4	Falsche BLOCK NUMBER
5	Falscher KEY MODE
6	Falscher OPERATION MODE
7	Nicht vorhandene PROGRAM NUMBER

9	BLOCK ist zu groß
10	Kein Speicher mehr frei
11	Falscher FUNCTION CODE
12	Diskettenfehler
13	STOP, EMERGENCY STOP
14	Nicht spezifiziert
15	Außerhalb des Arbeitsbereichs oder fehlender Startpunkt
16	Arbeitsstop
17	Geschwindigkeit ist Null
18	Kein Schweißroboter
30	Falsche Prüfsumme in den CONFIGURATION DATA
31	Fortfahren nicht möglich
32	Falsche Schweißdatenversion
33	Lesen eines LOCATION Registers mit aktiven TCP 20-29 nicht möglich
34	Bewegen des Roboters mit einem aktiven TCP 20-29 nicht möglich
35	Ändern der RESOLVER DATA nicht möglich
36	Wert des ANALOG OUTPUTS außerhalb des Wertebereichs
99	TIMEOUT
255	Nicht das gesamte Telegramm erhalten

8. Implementierung der ORI - Komponenten

8.1. Einleitung

Das vorige Kapitel beschäftigte sich mit dem Aufbau der Schichten und deren Schnittstellen, dieses Kapitel widmet sich der tatsächlichen Implementierung in DELPHI.

Die einzelnen Unterkapitel sind natürlich wieder nach Schichten getrennt.

8.2. RS232

Die RS232 – Komponente wurde mit Adaptierungen aus einem frei verfügbaren Modul übernommen.

8.2.1. Konfiguration

Die Konfiguration der seriellen Schnittstelle erfolgt über PROPERTIES der RS232 Komponente:

- property ComPort: TComPortNumber read FComPort write SetComPort default pnCOM1;
- property ComPortSpeed: TComPortBaudRate read FcomPortBaudRate write SetComPortBaudRate default br9600;
- property ComPortDataBits: TComPortDataBits read FcomPortDataBits write SetComPortDataBits default db8BITS;
- property ComPortStopBits: TComPortStopBits read FcomPortStopBits write SetComPortStopBits default sb1BITS;
- property ComPortParity: TComPortParity read FcomPortParity write SetComPortParity default ptEVEN;
- property ComPortHwHandshaking: TcomPortHwHandshaking read FComPortHwHandshaking write SetComPortHwHandshaking default hhNONE;
- property ComPortSwHandshaking: TcomPortSwHandshaking read FComPortSwHandshaking write SetComPortSwHandshaking default shNONE;
- property ComPortInBufSize: word read FcomPortInBufSize write SetComPortInBufSize default 2048;
- property ComPortOutBufSize: word read FcomPortOutBufSize write SetComPortOutBufSize default 2048;
- property ComPortPollingDelay: word read FcomPortPollingDelay write SetComPortPollingDelay default 500;
- property TimeOut: integer read FTimeOut write FTimeOut default 10;

COMPORT gibt den seriellen Port des Computers an, der verwendet werden soll. COMPORTSPEED ist die Schnittstellengeschwindigkeit. Für den ABB Roboter ist hier 9600 Bit/s zu wählen. COMPORTDATABITS gibt die Anzahl der Datenbits an. Hier ist 8 Bit einzustellen. COMPORTSTOPBITS ist für das Stopbit zuständig, es wird 1 Stopbit

verwendet. Die Parität wird mit `COMPORTPARITY` festgelegt und ist auf gerade zu stellen. Da keine Software - oder Hardware – Flußkontrolle benutzt wird, sind `COMPORTHWHANDSHAKING` und `COMPORTSWHANDSHAKING` nicht zu aktivieren. Die Puffergröße für das Senden bzw. Empfangen von Daten wird mittels `COMPORTINBUFSIZE` und `COMPORTOUTBUFSIZE` festgelegt. Der Standardwert ist 2048 Byte. `COMPORTPOLLINGDELAY` ist das Intervall in [ms] zwischen zwei Tests, ob Daten vom Roboter empfangen wurden. Normalerweise ist hier eine halbe Sekunde eingetragen. Zuletzt kann man mit der `PROPERTY TIMEOUT` in [s] festlegen, wie lange auf eine Antwort vom Roboter gewartet werden soll. Es sind 10 Sekunden eingestellt. Diese `PROPERTY` bewirkt bei zu kleinem Wert eine Fehlermeldung durch Zeitablauf obwohl eventuell kein Fehler aufgetreten ist, sondern die Operation beim Roboter länger gedauert hat, dies ist z.B. bei langsamen, langen Bewegungen so. Andererseits muß bei zu hohem Wert lange auf einen Zeitablauf gewartet werden.

Alle diese `PROPERTIES` können nicht nur im `OBJECT INSPECTOR` eingestellt werden, sondern auch zur Laufzeit geändert werden.

8.2.2. Schnittstelle zu ADLP10

Als Schnittstelle zum darüberliegenden ADLP10 – Protokoll dienen zwei Funktionen:

- `function SendString(aStr: string): boolean;`
- `function ReadString(var aResStr: string; aCount: word): boolean;`

Die Funktion `SENDSTRING` dient zum Senden einer Zeichenkette `ASTR` über die serielle Schnittstelle an den Kommunikationspartner. `READSTRING` liest im umgekehrten Fall `ACOUNT` Zeichen von der Schnittstelle in die Variable `ARESSTR` ein. Der Rückgabewert der Funktionen bestimmt, ob sie ordnungsgemäß ausgeführt wurden (`TRUE`) oder ob Fehler aufgetreten sind (`FALSE`).

8.2.3. Der Timer

Als zusätzliche Funktionalität löst die RS232 - Komponente einen Event aus, falls Daten vom Roboter im Empfangspuffer eingelangt sind. Dies wird durch den Timer angestoßen und nur dann an die oberen Schichten weitergeleitet, wenn die `ARAP` - Schichte derzeit keine Funktion bearbeitet.

Notwendig ist dies, da auch der Roboter zu jeder Zeit an den Computer `COMMANDS`, wie z.B. `TEACHING` – Befehle oder eine `SPONTANEOUS MESSAGE` schicken kann.

Diese, von der funktionalen Schicht zu behandelnden Daten, werden bereits von der RS232 - Komponente erkannt, um weniger Kommunikation zwischen den Schichten zu erreichen. (Würde von der funktionalen Schicht in Intervallen überprüft, ob Daten eingelangt sind, so müßte sehr häufig über die Kommunikationsschicht die RS232 Komponente befragt werden.)

8.3.ADLP10

Das ADLP10 - Protokoll verwendet die im Kapitel „RS232“ vorgestellten Funktionen READSTRING und SENDSTRING und stellt der darüberliegenden Schicht die Funktionen SEND und RECEIVE zur Verfügung.

8.3.1. Konfiguration

Die ADLP10 Komponente hat keine Konfigurationsparameter.

8.3.2. Schnittstelle zu ARAP

Dem ARAP Protokoll stellt die ADLP10 Komponente zwei Funktionen zur Verfügung:

- function Receive(var aReceiveList: TList): boolean;
- function Send(aSendList: TList): boolean;

Beide Funktionen benutzen als Parameter eine Liste. Diese Liste enthält die empfangenen bzw. die zu versendenden Telegramme. Der Rückgabewert der Funktionen bestimmt, ob sie ordnungsgemäß ausgeführt wurden (TRUE) oder ob Fehler aufgetreten sind (FALSE).

8.4.ARAP

Das ARAP Protokoll stellt bereits jene Funktionen zur Verfügung, die Anwendungsprogramme verwenden sollen. Die gesamte Roboterfunktionalität wird in dieser Schicht implementiert.

8.4.1. Konfiguration

Auch die ARAP Komponente hat keine Konfigurationsparameter.

8.4.2. Schnittstelle zu Anwendungen bzw. RAL

ARAP stellt folgende Funktionen bereit:

- function SendProgram(aMode, aProgramNumber, aBlockNumber: integer; aProgramData: string): boolean;
- function StartProgram(aFirstInstruction, aProgramNumber: integer): boolean;
- function StopProgram: boolean;
- function ReadProgramStatus(var aProgramStatus: string): boolean;
- function DeleteProgram(aProgramNumber: integer): boolean;
- function DiskLoadProgram(aMode, aProgramNumber, aBlockNumber: integer): boolean;
- function ReceiveProgram(aMode, aProgramNumber, aBlockNumber: integer; var aProgramData: string): boolean;
- function ReadTCP(aRegister: integer; var aTCP: string): boolean;
- function ReadLocation(aRegister: integer; var aLocation: string): boolean;
- function ReadRegister(aRegister: integer; var aValue: string): boolean;
- function ReadSensor(aRegister: integer; var aSensor: string): boolean;
- function ReadInput(aPort: integer; var aValue: integer): boolean;
- function ReadOutput(aPort: integer; var aValue: integer): boolean;
- function ReadConfig(aBlockNumber: integer; var aConfig: string): boolean;
- function ReadFrame(aRegister: integer; var aFrame: string): boolean;
- function ReadStatus(var aStatus: string): boolean;
- function WriteTCP(aRegister: integer; aTCP: string): boolean;
- function ActiveTCP(aRegister: integer): boolean;
- function WriteLocation(aRegister: integer; aLocation: string): boolean;
- function WriteRegister(aRegister: integer; aValue: string): boolean;
- function WriteSensor(aRegister: integer; aSensor: string): boolean;
- function WriteOutput(aPort, aValue: integer): boolean;
- function WriteConfig(aBlockNumber: integer; aConfig: string): boolean;
- function WriteFrame(aRegister: integer; aFrame: string): boolean;
- function WriteMode(aMode: integer): boolean;
- function Move(aOrientation, aMode, aMove, aVelocity, aHandPos: integer; aVector: TRPoint): boolean;

- function ReturnHome: boolean;

Alle Funktionen liefern FALSE im Fehlerfall und TRUE bei gültiger Ausführung zurück.

Die Funktionen im Detail: (In Klammer ist der verwendete FUNCTION CODE angegeben.)

8.4.2.1. Funktion SENDPROGRAM (FC1)

Diese Funktion sendet ein Roboterprogramm vom Computer an den Roboter.

- function SendProgram(aMode, aProgramNumber, aBlockNumber: integer; aProgramData: string): boolean;

AMODE gibt an, ob ein Programm (Eins) oder ein Block (Null) gesendet wird. APROGRAMNUMBER ist die Programmnummer, sie ist aber nur relevant, wenn ein Programm gesendet wird. ABLOCKNUMBER gibt die Nummer des Blocks an. APROGRAMDATA ist das Roboterprogramm als STRING.

8.4.2.2. Funktion STARTPROGRAM (FC2)

Starten eines Roboterprogramms.

- function StartProgram(aFirstInstruction, aProgramNumber: integer): boolean;

Mit FIRSTINSTRUCTION wird angegeben, ob das Programm vom Beginn (Null) oder von der Position des letzten Halts gestartet werden soll (Eins). Die APROGRAMNUMBER gibt das zu startende Programm an, ist aber nur relevant, wenn FIRSTINSTRUCTION auf Null gesetzt ist.

8.4.2.3. Funktion STOPPROGRAM (FC3)

Hält ein laufendes Roboterprogramm an.

- function StopProgram: boolean;

Diese Funktion benötigt keine Parameter.

8.4.2.4. Funktion READPROGRAMSTATUS (FC21)

Hier wird das Auslesen des Programmspeichers ermöglicht.

- function ReadProgramStatus(var aProgramStatus: string): boolean;

In der Variable APROGRAMSTATUS wird die Information als STRING zurückgeliefert. Details über diese Informationen finden sich im Kapitel „FUNCTION CODE 21“.

8.4.2.5. Funktion DELETEDPROGRAM (FC22)

Mit dieser Funktion kann ein geladenes Programm aus dem Roboterspeicher entfernt werden.

- function DeleteProgram(aProgramNumber: integer): boolean;

APROGRAMNUMBER gibt das zu löschende Programm an.

8.4.2.6. Funktion DISKLOADPROGRAM (FC23)

Laden eines Roboterprogramms von der Diskettenstation der Steuerung in den Roboterspeicher.

- function DiskLoadProgram(aMode, aProgramNumber, aBlockNumber: integer): boolean;

AMODE gibt an, ob ein Programm (Eins) oder ein Block (Null) gelesen werden soll. APROGRAMNUMBER ist die Programmnummer, sie ist aber nur relevant, wenn ein Programm von Diskette gelesen wird. ABLOCKNUMBER gibt die Nummer des Blocks an.

8.4.2.7. Funktion RECEIVEPROGRAM (FC29)

Dient zum Empfangen eines Roboterprogramms aus dem Roboterspeicher.

- function ReceiveProgram(aMode, aProgramNumber, aBlockNumber: integer; var aProgramData: string): boolean;

AMODE gibt an, ob ein Programm (Eins) oder ein Block (Null) empfangen werden soll. APROGRAMNUMBER ist die Programmnummer, sie ist aber nur relevant, wenn ein Programm empfangen wird. ABLOCKNUMBER gibt die Nummer des Blocks an. APROGRAMDATA ist das Roboterprogramm als STRING.

8.4.2.8. Funktion READTCP (FC4)

Liest den Wert eines TCP Registers in der Robotersteuerung.

- function ReadTCP(aRegister: integer; var aTCP: string): boolean;

AREGISTER gibt das TCP Register an, ATCP liefert den Wert als STRING zurück.

8.4.2.9. Funktion READLOCATION (FC5)

Liest den Wert eines LOCATION Registers in der Robotersteuerung.

- function ReadLocation(aRegister: integer; var aLocation: string): boolean;

AREGISTER gibt das LOCATION Register an, ALOCATION liefert den Wert als STRING zurück.

8.4.2.10. Funktion READREGISTER (FC6)

Liest den Wert eines DATA Registers in der Robotersteuerung.

- function ReadRegister(aRegister: integer; var aValue: string): boolean;

AREGISTER gibt das DATA Register an, AVALUE liefert den Wert als STRING zurück.

8.4.2.11. Funktion READSENSOR (FC7)

Liest den Wert eines SENSOR Registers in der Robotersteuerung.

- function ReadSensor(aRegister: integer; var aSensor: string): boolean;

AREGISTER gibt das SENSOR Register an, ASENSOR liefert den Wert als STRING zurück.

8.4.2.12. Funktion READINPUT (FC8)

Liest den Wert eines DIGITAL INPUTS in der Robotersteuerung.

- function ReadInput(aPort: integer; var aValue: integer): boolean;

APORT gibt den DIGITAL INPUT an, AVALUE liefert den Wert als Zahl zurück.

8.4.2.13. Funktion READOUTPUT (FC9)

Liest den Wert eines DIGITAL OUTPUTS in der Robotersteuerung.

- function ReadOutput(aPort: integer; var aValue: integer): boolean;

APORT gibt den DIGITAL OUTPUT an, AVALUE liefert den Wert als Zahl zurück.

8.4.2.14. Funktion READCONFIG (FC10)

Liest die CONFIGURATION DATA in der Robotersteuerung.

- function ReadConfig(aBlockNumber: integer; var aConfig: string): boolean;

ABLOCKNUMBER gibt die Nummer des Blocks an. ACONFIG liefert die CONFIGURATION DATA als STRING zurück.

8.4.2.15. Funktion READFRAME (FC11)

Liest den Wert eines FRAME Registers in der Robotersteuerung.

- function ReadFrame(aRegister: integer; var aFrame: string): boolean;

AREGISTER gibt das FRAME Register an, AFRAME liefert den Wert als STRING zurück.

8.4.2.16. Funktion READSTATUS (FC19)

Diese Funktion liest verschiedene Roboterstatus - Informationen aus der Steuerung.

- function ReadStatus(var aStatus: string): boolean;

ASTATUS liefert den Roboterstatus als STRING zurück.

Details über diese Informationen finden sich im Kapitel „FUNCTION CODE 19“.

8.4.2.17. Funktion WRITETCP (FC12)

Schreibt den Wert eines TCP Registers in der Robotersteuerung.

- function WriteTCP(aRegister: integer; aTCP: string): boolean;

AREGISTER gibt das TCP Register an, ATCP bestimmt den Wert.

8.4.2.18. Funktion ACTIVETCP (FC?)

Aktiviert einen TCP in der Robotersteuerung.

DIESE FUNKTION KONNTE NOCH NICHT IMPLEMENTIERT WERDEN, DA ES DAFÜR KEINE BEKANNTE ARAP - FUNKTION GIBT. DER TCP MUß DAHER HÄNDISCH MIT DER PROGRAMMING UNIT GESETZT WERDEN.

- function ActiveTCP(aRegister: integer): boolean;

AREGISTER gibt das TCP Register an.

8.4.2.19. Funktion WRITELOCATION (FC13)

Schreibt den Wert eines LOCATION Registers in der Robotersteuerung.

- function WriteLocation(aRegister: integer; aLocation: string): boolean;

AREGISTER gibt das LOCATION Register an, ALOCATION bestimmt den Wert.

8.4.2.20. Funktion WRITEREGISTER (FC14)

Schreibt den Wert eines DATA Registers in der Robotersteuerung.

- function WriteRegister(aRegister: integer; aValue: string): boolean;

AREGISTER gibt das DATA Register an, AVALUE bestimmt den Wert.

8.4.2.21. Funktion WRITESENSOR (FC15)

Schreibt den Wert eines SENSOR Registers in der Robotersteuerung.

- function WriteSensor(aRegister: integer; aSensor: string):boolean;

AREGISTER gibt das SENSOR Register an, ASENSOR bestimmt den Wert.

8.4.2.22. Funktion WRITEOUTPUT (FC16)

Schreibt den Wert eines DIGITAL OUTPUTS in der Robotersteuerung.

- function WriteOutput(aPort, aValue: integer): boolean;

APOINT gibt den DIGITAL OUTPUT an, AVALUE bestimmt den Wert.

8.4.2.23. Funktion WRITECONFIG (FC17)

Schreibt die CONFIGURATION DATA in der Robotersteuerung.

- function WriteConfig(aBlockNumber: integer; aConfig: string): boolean;

ABLOCKNUMBER gibt die Nummer des Blocks an. ACONFIG setzt die Konfiguration.

8.4.2.24. Funktion WRITEFRAME (FC18)

Schreibt den Wert eines FRAME Registers in der Robotersteuerung.

- `function WriteFrame(aRegister: integer; aFrame: string): boolean;`

AREGISTER gibt das FRAME Register an, AFRAME bestimmt den Wert.

8.4.2.25. Funktion WRITEMODE (FC20)

Setzt den Roboter in einen bestimmten Modus.

- `function WriteMode(aMode: integer): boolean;`

AMODE bestimmt den Robotermodus. Bei Null ist er auf STAND BY, bei Eins auf OPERATION, bei Zwei auf SYNCHRONIZATION und bei Drei auf SYNCHRONIZATION AND OPERATION gesetzt.

8.4.2.26. Funktion MOVE (FC24)

Bewegt den Roboter.

- `function Move(aOrientation, aMode, aMove, aVelocity, aHandPos: integer; aVector: TRPoint): boolean;`

AORIENTATION bestimmt Handgelenkorientierung (Null) oder Werkzeugorientierung (Eins). AMODE ist für absolute (Null) bzw. relative (Eins) Bewegung. AMOVE setzt rechtwinkelige (Null) oder Roboter (Eins) – Koordinaten. AVELOCITY bestimmt die Geschwindigkeit der Bewegung in [mm/s]. HANDPOS ist Null.

In AVECTOR wird die zu erreichende Endposition angegeben. Siehe dazu das Kapitel „Spezielle Datenstrukturen“.

8.4.2.27. Funktion RETURNHOME (FC24)

Bewegt den Roboter in die Ausgangsposition.

- `function ReturnHome: boolean;`

Es wird die Funktion MOVE mit Standardparametern aufgerufen.

8.5. Spezielle Datenstrukturen

8.5.1. TRPOINT

Die Definition von TRPOINT:

```
type
TRPoint = class(TObject)
private
FX: real;
FY: real;
FZ: real;
FQ1: real;
FQ2: real;
FQ3: real;
FQ4: real;

procedure SetX(const Value: real);
procedure SetY(const Value: real);
procedure SetZ(const Value: real);
procedure SetQ1(const Value: real);
procedure SetQ2(const Value: real);
procedure SetQ3(const Value: real);
procedure SetQ4(const Value: real);
public
constructor Create;

property X: real read FX write SetX;
property Y: real read FY write SetY;
property Z: real read FZ write SetZ;
property Q1: real read FQ1 write SetQ1;
property Q2: real read FQ2 write SetQ2;
property Q3: real read FQ3 write SetQ3;
property Q4: real read FQ4 write SetQ4;
end;
```

X, Y und Z sind die Ziel - Koordinaten für den aktuellen TCP. Q1 – Q4 geben die Orientierung des Roboters zu diesem Punkt in Quaternion Notation an.

8.6.EXCEPTIONS

Tritt in einer Funktion einer Komponente ein Fehler auf, so wird eine EXCEPTION ausgelöst. Hier ein Beispiel aus der RS232 - Komponente:

```
function TCommPortDriver.SendString( aStr: string ): boolean;
begin
  [...]
  Result:=False;
  Exception.CreateHelp('RS232.SendString: Send not possible !', 102);
end;
```

Kann die Funktion SENDSTRING nicht ordnungsgemäß ausgeführt werden, wird eine EXCEPTION ausgelöst. Diese kann in einer der darüberliegenden Schichten abgefangen werden, um darauf zu reagieren. Zusätzlich wird natürlich auch der Rückgabewert der Funktion FALSE.

In diesem Beispiel wird die EXCEPTION im ARAP – Protokoll abgefangen:

```
try
  RCL.Send(SendList);
  [...]
except
  on E:Exception do
  begin
    [...]
    Busy := false;
    raise Exception.CreateHelp(E.Message, E.HelpContext);
  end;
end;
```

Damit kann das ARAP – Protokoll auch im Fehlerfall zuvor aufgebaute Datenstrukturen wieder löschen und z.B. auch das BUSY PROPERTY wieder auf FALSE setzen. Gleich danach wird aber eine neuerliche EXCEPTION ausgelöst, um auch in die oberen Schichten den Fehler weiterzugeben. Auch diese Schichten können dann ihrerseits eine Fehlerbehandlung durchführen.

Wird diese EXCEPTION nicht abgefangen, erhält der Anwender eine Meldung:



9. Beispiel – Anwendung ORIDEMO

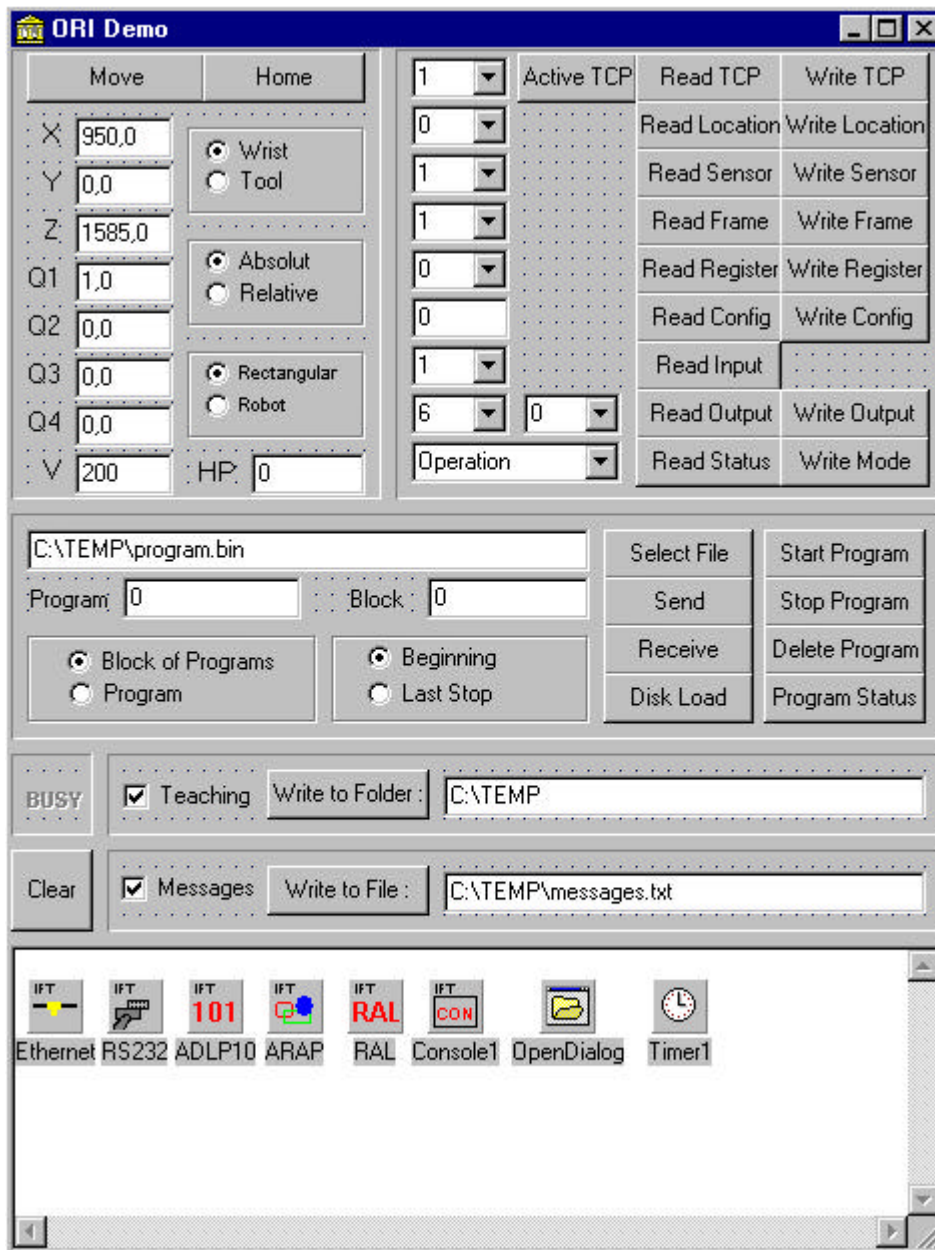
9.1. Einleitung

In diesem Kapitel wird die Beispiel – Anwendung ORIDEMO vorgestellt. Sie soll Anwendern der ORI – Architektur als Muster für eigene Entwicklungen dienen.

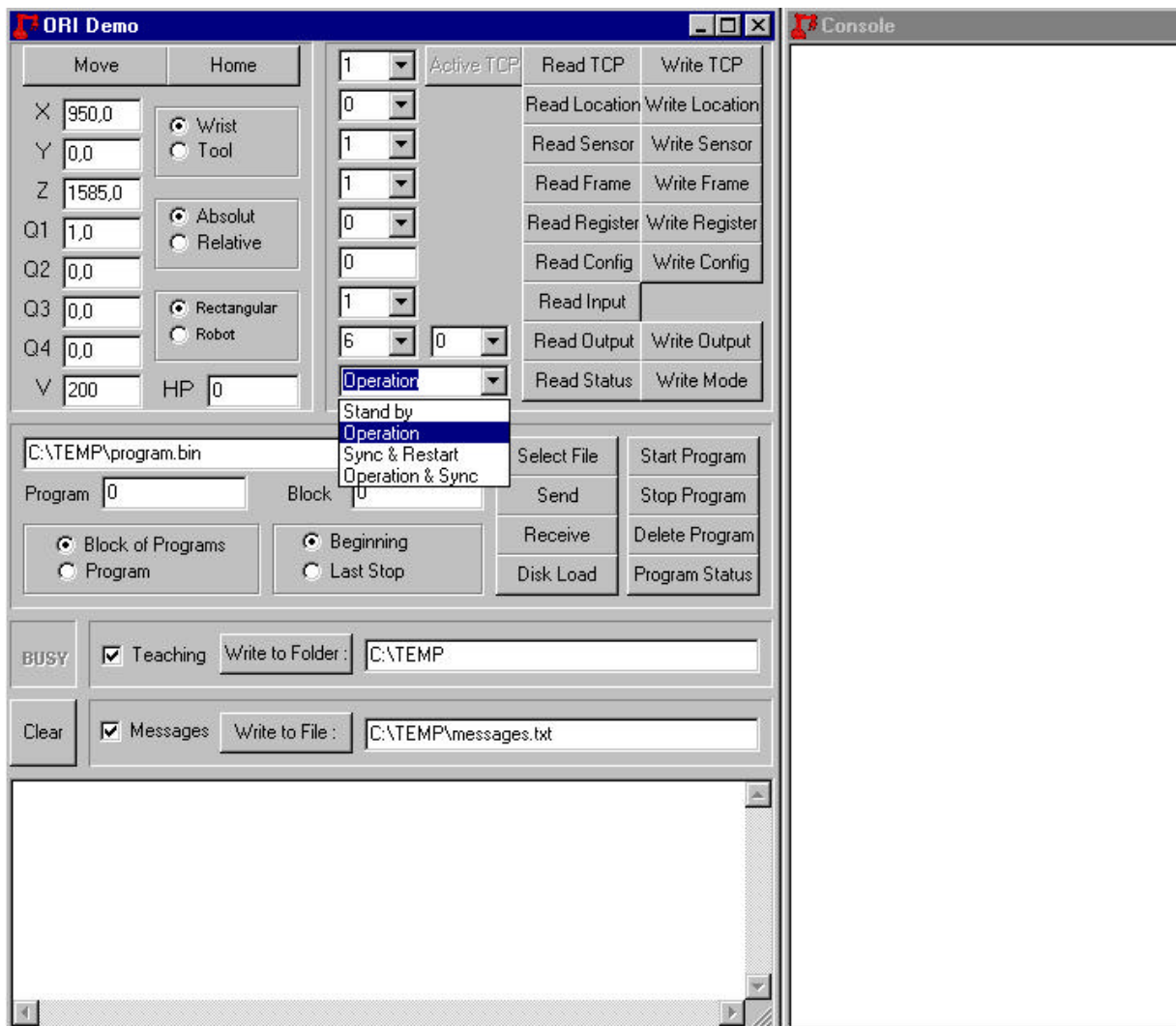
Ein genaues Listing befindet sich im Anhang.

9.2. Oberfläche

Als Beispiel wurde eine Anwendung programmiert, von der aus alle Funktionen der RAL – Komponente getestet werden können.



Im linken oberen Bereich kann der Roboter bewegt werden. Rechts daneben können die Register der Steuerung gelesen und geschrieben werden. Im mittleren Bereich ist das Senden, Empfangen, Starten und Stoppen von Programmen in der Steuerung sichtbar. Darunter befindet sich die Option, das TEACHING auszuschalten. Es kann ein Folder für das TEACHINGFILE und darunter eine Datei für die MESSAGES angegeben werden. Auch die MESSAGES können deaktiviert werden. Im untersten Bereich ist ein MEMO zu sehen, welches verschiedene Meldungen von ORIDEMO ausgibt. In diesem MEMO sind auch die verwendeten Komponenten gut sichtbar. Neben den TEACHING – Einstellungen ist auch das BUSY – FLAG sichtbar, welches durch Rotfärbung Aktivität in der ARAP – Komponente anzeigt.

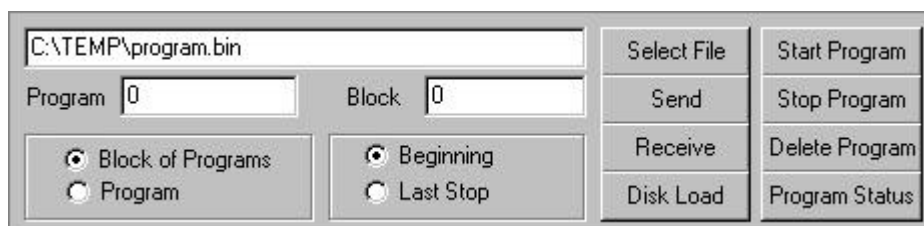


Im laufenden Zustand ist rechts daneben auch eine Konsole sichtbar, auf die aus allen Komponenten Informationen ausgegeben werden können.

9.3. Programmierdetails

9.3.1. Aufruf einer RAL – Funktion

Die Schaltfläche START PROGRAM startet z.B. ein Programm in der Steuerung.



Für die Funktion

- `function StartProgram(aFirstInstruction, aProgramNumber: integer): boolean;`

werden die Eingabefelder PROGRAM und die Radioknöpfe BEGINNING/LAST STOP verwendet. Der dazu gehörige Programmteil sieht so aus:

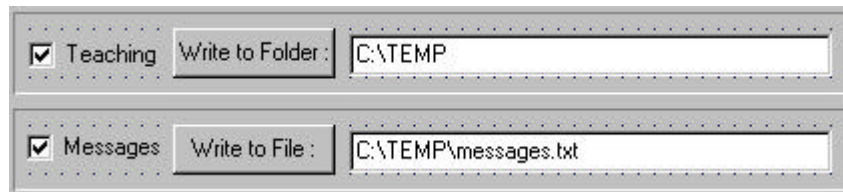
```

procedure TFormDemo.ButtonStartProgramClick(Sender: TObject);
var FirstInstruction: integer;
begin
    if RadioButtonBeginning.Checked=true then FirstInstruction:=0
        else FirstInstruction:=1;
    if not RAL.StartProgram(FirstInstruction, StrToInt(EditProgramNumber.Text))
        then MemoORIDemo.Lines.Add('Start Program not possible !')
        else MemoORIDemo.Lines.Add('Start Program successfully !');
end;

```

9.3.2. TEACHING und MESSAGES

TEACHING und MESSAGES sind PROPERTIES der ARAP – Komponente. Sie sind im PUBLISHED Bereich definiert und daher auch über den OBJECT INSPECTOR einstellbar. Sie können aber auch zur Laufzeit geändert werden.



Im Listing sieht man die einfache Verwendung:

```

procedure TFormDemo.TeachingClick(Sender: TObject);
begin
    ARAP.Teaching:=CheckBoxTeaching.Checked;
    ARAP.TeachingFolder:=EditTeaching.Text;
    MemoORIDemo.Lines.Add('Change teaching successfully !');
end;

```

```
end;
```

```
procedure TFormDemo.MessagesClick(Sender: TObject);
```

```
begin
```

```
  ARAP.Messages:=CheckBoxMessages.Checked;
```

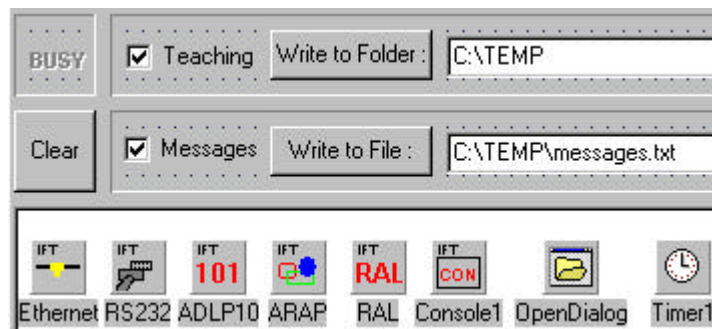
```
  ARAP.MessagesFile:=EditMessages.Text;
```

```
  MemoORIDemo.Lines.Add('Change messages successfully !');
```

```
end;
```

9.3.3. BUSY

Auf der Form liegt neben den ORI – Komponenten auch ein TIMER. Das INTERVAL ist auf 50 [ms] gesetzt.



Der Programmteil für das BUSY – FLAG lautet:

```
procedure TFormDemo.Timer1Timer(Sender: TObject);
```

```
begin
```

```
  if RAL.Busy then LabelBusy.Enabled:=true else LabelBusy.Enabled:=false;
```

```
end;
```

Auch BUSY ist eine PROPERTY der RAL – Komponente.

10. Schlußwort

Die ORI – ARCHITECTURE ist der erste Schritt in die moderne Programmieretechnik auf dem Gebiet der Roboter. Es wird erstmalig objektorientierte, grafische Programmierung auf diesem Sektor eingeführt. Die Trennung von komplexen Aufgaben in abgeschlossene Schichten mit genau definierten Schnittstellen erleichtert nicht nur das Verständnis der Implementierung, sondern ist für die Wiederverwendbarkeit und den Austausch von Komponenten zwingend notwendig.

Diese Arbeit beschränkt sich auf die Steuerung von Robotern, die ORI – ARCHITECTURE ist aber prinzipiell für alle Maschinen verwendbar. Auch die Verteilung verschiedener Aufgaben (Schichten) auf unterschiedliche Computer ist leicht möglich. Da moderne Programmieretechnik auch oder gerade auf dem Gebiet der Maschinensteuerung notwendig ist, wird die Idee „ORI“ sicher weiterentwickelt werden.

11. Anhang

11.1. FUNCTION CODES

11.1.1. FUNCTION CODE 1: Roboterprogramm senden

Mit diesem Befehl wird ein Programm vom Computer an den Roboter geschickt.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=12+Programm								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=1								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0/1	0	0	1	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER								8
								9
BLOCK NUMBER								10
								11
ROBOT PROGRAM								12
								...
								NOB-1

FUNCTION SUFFIX:

Null bedeutet BLOCK OF PROGRAMS, das ist ein Block mit mehreren Programmen und Eins bedeutet PROGRAM.

PROGRAM NUMBER:

Nummer des Programms, 0-9999. Nur relevant, wenn die FUNCTION SUFFIX Eins ist.

BLOCK NUMBER:

Nummer des Blocks, 0-9999.

ROBOT PROGRAM:

Roboterprogramm.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=12 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=1								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER (ERRORCODE)								8
								9
BLOCK NUMBER								10
								11

11.1.2. FUNCTION CODE 2: Programm starten

Dieser Befehl startet ein Roboterprogramm.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=10								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=2								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER								8
								9

FUNCTION SUFFIX:

Null bedeutet START FROM BEGINNING und Eins bedeutet START FROM LAST STOP.

PROGRAM NUMBER:

Nummer des Programms, 0-9999. Nur relevant, wenn die FUNCTION SUFFIX Null ist.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=10								0
								1

DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=2								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER (ERRORCODE)								8
								9

11.1.3. FUNCTION CODE 3: Programm stoppen

Dies ist der Befehl um ein laufendes Roboterprogramm zu stoppen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=3								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Nicht relevant.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=3								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
(ERRORCODE)								8
								9

11.1.4. FUNCTION CODE 4: TCP Register lesen

Dies ist der Befehl um den Wert eines TCP Registers zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=4								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6

	7
--	---

FUNCTION SUFFIX:

Hier wird das TCP Register angegeben, 0-29.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=20 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=4								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
TCP REGISTER (ERRORCODE)								8
								...
								19

TCP REGISTER:

Wert des TCP Registers, 12 Byte.

11.1.5. FUNCTION CODE 5: LOCATION Register lesen

Dies ist der Befehl um den Wert eines LOCATION Registers zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=5								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Hier wird das LOCATION Register angegeben, 0-199.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=48 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=5								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
LOCATION REGISTER (ERRORCODE)								8
								...

	19
--	-----------

LOCATION REGISTER:

Wert des LOCATION Registers, 40 Byte.

11.1.6. FUNCTION CODE 6: Register DATA lesen

Dies ist der Befehl um die Register DATA zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=6								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Hier wird das Register angegeben, 0-119.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=10								0
								1

DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=6								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
REGISTER (ERRORCODE)								8
								9

REGISTER:

Wert des Registers, 2 Byte.

11.1.7. FUNCTION CODE 7: SENSOR Register lesen

Dies ist der Befehl um den Wert eines SENSOR Registers zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=7								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Hier wird das SENSOR Register angegeben, 1-16.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=18 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=7								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
SENSOR REGISTER (ERRORCODE)								8
								...
								17

SENSOR REGISTER:

Wert des SENSOR Registers, 10 Byte.

11.1.8. FUNCTION CODE 8: DIGITAL INPUTS lesen

Dies ist der Befehl um den Wert eines DIGITAL INPUTS zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0

								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=8								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Hier wird der DIGITAL INPUT angegeben, 0-190.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=9 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=8								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
DIGITAL INPUT (ERRORCODE)								8
								9

DIGITAL INPUT:

Wert des DIGITAL INPUTS, 1 Byte.

11.1.9. FUNCTION CODE 9: DIGITAL OUTPUTS lesen

Dies ist der Befehl um den Wert eines DIGITAL OUTPUT zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=9								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Hier wird der DIGITAL OUTPUT angegeben, 0-190.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=9 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3

FUNCTION CODE=9								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
DIGITAL OUTPUT (ERRORCODE)								8
								9

DIGITAL OUTPUT:

Wert des DIGITAL OUTPUT, 1 Byte.

11.1.10. FUNCTION CODE 10: CONFIGURATION DATA lesen

Dies ist der Befehl um die CONFIGURATION DATA zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=10								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
BLOCK NUMBER								8
								9

FUNCTION SUFFIX:

Nicht relevant.

BLOCK NUMBER:

Nummer des Programmblocks, 0-9999.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=10								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
BLOCK NUMBER (ERRORCODE)								8
								9
CONFIGURATION DATA								10
								...
								NOB-1

CONFIGURATION DATA:

Gelesene CONFIGURATION DATA.

11.1.11. FUNCTION CODE 11: FRAME Register lesen

Dies ist der Befehl um den Wert eines FRAME Registers zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=11								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Hier wird das FRAME Register angegeben, 0-5.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=22 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=11								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6

	7
FRAME REGISTER (ERRORCODE)	8
	...
	21

FRAME REGISTER:

Wert des FRAME Registers, 14 Byte.

11.1.12. FUNCTION CODE 12: TCP Register schreiben

Dies ist der Befehl um den Wert eines TCP Registers zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=20								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=12								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
TCP REGISTER								8
								...
								19

FUNCTION SUFFIX:

Hier wird das TCP Register angegeben, 1-29.

TCP REGISTER:

Wert des TCP Registers, 12 Byte.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=12								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
(ERRORCODE)								8
								9

11.1.13. FUNCTION CODE 13: LOCATION Register schreiben

Dies ist der Befehl um den Wert eines LOCATION Registers zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=48								0
								1

DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=13								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
LOCATION REGISTER								8
								...
								47

FUNCTION SUFFIX:

Hier wird das LOCATION Register angegeben, 0-199.

LOCATION REGISTER:

Wert des LOCATION Registers, 40 Byte.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=13								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6

	7
(ERRORCODE)	8
	9

11.1.14. FUNCTION CODE 14: Register DATA schreiben

Dies ist der Befehl um den Wert eines Registers zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=10								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=14								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
REGISTER								8
								9

FUNCTION SUFFIX:

Hier wird das Register angegeben, 0-119.

REGISTER:

Wert des Registers, 2 Byte.

FUNCTION SUFFIX	6
	7
SENSOR DATA	8
	...
	17

FUNCTION SUFFIX:

Hier wird der SENSOR angegeben, 1-16.

SENSOR DATA:

Wert des SENSORS, 10 Byte.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=15								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
(ERRORCODE)								8
								9

11.1.16. FUNCTION CODE 16: DIGITAL OUTPUTS schreiben

Dies ist der Befehl um einen DIGITAL OUTPUT zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=9								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=16								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
DATA								8

FUNCTION SUFFIX:

Hier wird der DIGITAL OUTPUT angegeben, 1-190.

DATA:

Wert des DIGITAL OUTPUTS, 1 Byte.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3

FUNCTION CODE=16								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
(ERRORCODE)								8
								9

11.1.17. FUNCTION CODE 17: CONFIGURATION DATA schreiben

Dies ist der Befehl um die CONFIGURATION DATA zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=17								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0/1	0	0	1	
FUNCTION SUFFIX								6
								7
BLOCK NUMBER								8
								9
CONFIGURATION DATA								10
								...
								NOB-1

FUNCTION SUFFIX:

Nicht relevant.

BLOCK NUMBER:

Nummer des Programmblocks, 0-9999.

CONFIGURATION DATA:

Zu schreibende CONFIGURATION DATA.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=10								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=17								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
BLOCK NUMBER (ERRORCODE)								8
								9

11.1.18. FUNCTION CODE 18: FRAME Register schreiben

Dies ist der Befehl um den Wert eines FRAME Registers zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=22								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=18								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
FRAME REGISTER								8
								...
								21

FUNCTION SUFFIX:

Hier wird das FRAME Register angegeben, 1-5.

FRAME REGISTER:

Wert des FRAME Registers, 14 Byte.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3

FUNCTION CODE=18								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
(ERRORCODE)								8
								9

11.1.19. FUNCTION CODE 19: STATUS lesen

Dies ist der Befehl um den Roboter STATUS zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=19								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Nicht relevant.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=60								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=19								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0/1	0	0	1	0	
FUNCTION SUFFIX								6
								7
DUMMY								8
								9
								10
PROGRAM NUMBER								11
								12
INSTRUCTION NUMBER								13
								14
ACTUAL TCL								15
ACTUAL FRAME								16
LR	IR	PU	KEY	M	O	D	E	17
COORDINATES								18
								...
								59

ORT – ORIENTATION:

Bei auf Null gesetztem Bit werden handgelenkorientierte Koordinaten verwendet, bei Eins werkzeugorientierte Koordinaten.

DUMMY:

Nicht relevant.

PROGRAM NUMBER:

Aktives Programm.

INSTRUCTION NUMBER:

Aktive INSTRUCTION.

ACTUAL TCP:

Aktiver TCP.

ACTUAL FRAME:

Aktiver FRAME.

LR – LOCAL/REMOTE:

LR	Erklärung
0	LOCAL
1	REMOTE

IR – INTERRUPT:

IR	Erklärung
0	PERMITTED

1	NOT PERMITTED
---	---------------

PU – PROGRAMMING UNIT:

PU	Erklärung
0	CONNECTED
1	NOT CONNECTED

KEY (Schlüsselschalter):

KEY	Erklärung
0	AUTO
1	AUTO oder TEST 100%

MODE:

MODE	Erklärung
0	STAND BY
1	OPERATION
2	EXECUTION
3	EMERGENCY STOP

COORDINATES:

Siehe Kapitel „Koordinaten“.

11.1.20. FUNCTION CODE 20: MODE schreiben

Dies ist der Befehl um den Roboter MODE zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=20								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

FUNCTION SUFFIX	Erklärung
0	STAND BY
1	OPERATION
2	SYNCHRONIZATION
3	OPERATION AND SYNCHRONIZATION

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3

FUNCTION CODE=20								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
(ERRORCODE)								8
								9

11.1.21. FUNCTION CODE 21: PROGRAM STATUS lesen

Dies ist der Befehl um den Roboter PROGRAM STATUS zu lesen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=21								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Nicht relevant.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=21								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0/1	0/1	1	0	
FUNCTION SUFFIX								6
								7
MEMORY (ERRORCODE)								8
								9
PROGRAM NUMBER 1								10
								11
PROGRAM NUMBER 2								12
								13
PROGRAM NUMBER N								...
								NOB-1

MEMORY:

Verfügbarer Speicher in Bytes.

PROGRAM NUMBER:

Programm im Roboter Speicher.

11.1.22. FUNCTION CODE 22: Programm löschen

Dies ist der Befehl um ein Roboter Programm aus der Steuerung zu löschen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=22								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7

FUNCTION SUFFIX:

Programmnummer, 0-9999.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=22								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6

	7
(ERRORCODE)	8
	9

11.1.23. FUNCTION CODE 23: Programm von Diskette laden

Dies ist der Befehl, um ein Roboter Programm von Diskette in die Steuerung zu laden.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=12								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=23								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER								8
								9
BLOCK NUMBER								10
								11

FUNCTION SUFFIX:

Null bedeutet BLOCK OF PROGRAMS, das ist ein Block mit mehreren Programmen und Eins bedeutet PROGRAM.

PROGRAM NUMBER:

Nummer des Programms, 0-9999. Nur relevant, wenn die FUNCTION SUFFIX Eins ist.

BLOCK NUMBER:

Nummer des Blocks auf der Diskette, 0-9999.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=12 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=23								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER (ERRORCODE)								8
								9
BLOCK NUMBER								10
								11

11.1.24. FUNCTION CODE 24: Bewegung

Dies ist der Befehl, um den Roboter zu bewegen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=60								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=24								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0/1	0	0	0	1	
FUNCTION SUFFIX=0-3								6
								7
PROGRAM NUMBER								8
								9
INSTRUCTION NUMBER								10
								11
HANDPOS								12
MOVE DATA								13
VELOCITY								14
								15
VPROG								16
								17
COORDINATES								18
								...
								59

ORT – ORIENTATION:

Bei auf Null gesetztem Bit werden handgelenkorientierte Koordinaten verwendet, bei Eins werkzeugorientierte Koordinaten.

FUNCTION SUFFIX:

FUNCTION SUFFIX	Erklärung
0	Absolute Bewegung
1	Relative Bewegung
2	Startpunkt
3	Endpunkt

PROGRAM NUMBER:

Nummer des zu aktivierenden Programms, 0-9999. Nicht notwendig.

INSTRUCTION NUMBER:

INSTRUCTION in diesem Programm, 0-65535. Nicht notwendig.

HANDPOS:

Null.

MOVE DATA:

MOVE DATA	Erklärung
0	Rechtwinkelige Koordinaten
2	Startpunkt
4	Roboterkoordinaten

VELOCITY:

Geschwindigkeit in [mm/s].

VPROG:

Geschwindigkeit in [%].

Die resultierende Roboter Geschwindigkeit ist das Produkt aus VELOCITY und VPROG.

COORDINATES:

Siehe Kapitel „Koordinaten“.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=12 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=24								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX=0-3								6
								7
PROGRAM NUMBER (ERRORCODE)								8
								9
INSTRUCTION NUMBER								10
								11

11.1.25. FUNCTION CODE 29: Roboter Programm empfangen

Dies ist der Befehl, um ein Roboter Programm vom Roboter zu empfangen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=12								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=29								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER								8
								9
BLOCK NUMBER								10
								11

FUNCTION SUFFIX:

Null bedeutet BLOCK OF PROGRAMS, das ist ein Block mit mehreren Programmen und Eins bedeutet PROGRAM.

PROGRAM NUMBER:

Nummer des Programms, 0-9999.

BLOCK NUMBER:

Nummer des BLOCKS, 0-9999.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=29								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0/1	0/1	1	0	
FUNCTION SUFFIX								6
								7
PROGRAM NUMBER (ERRORCODE)								8
								9
BLOCK NUMBER								10
								11
ROBOT PROGRAM								10
								...
								NOB-1

ROBOT PROGRAM:

Roboter Programm.

11.1.26. FUNCTION CODE 72: TEACHING

11.1.26.1. OPEN FILE

Dies ist der Befehl, um ein File für TEACHING - Koordinaten am Computer zu öffnen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=18								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX=1								6
								7
FILE NAME								8
								...
								17

FILE NAME:

Name für das File, 10 Byte im ASCII - CODE.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	

FUNCTION SUFFIX=1	6
	7
(ERRORCODE)	8
	9

11.1.26.2. CLOSE FILE

Dies ist der Befehl, um das offene File für TEACHING - Koordinaten am Computer zu schließen.

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX=2								6
								7

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5

0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX=2								6
								7
(ERRORCODE)								8
								9

11.1.26.3. DEFINE POSITION

Dies ist der Befehl, um eine Position in das File für TEACHING Koordinaten am Computer zu schreiben.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=68								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX=3								6
								7
POSITION NAME								8
								...
								23
DUMMY								24
HANDPOS								25
COORDINATES								26
								...

	67
--	-----------

POSITION NAME:

Name für die Position, 16 Byte in ASCII - CODE.

DUMMY:

Null. Nicht relevant.

HANDPOS:

Null.

COORDINATES:

Siehe Kapitel „Koordinaten“.

RESPONSE:

7	6	5	4	3	2	1	0		BIT / BYTE
NOB=8 (10 bei ERRORCODE)									0
									1
DESTINATION ADDRESS									2
SOURCE ADDRESS									3
FUNCTION CODE=72									4
-	-	-	ORT	MLI	RS	T	T		5
0	0	0	0	0	0/1	1	0		
FUNCTION SUFFIX=3									6
									7
(ERRORCODE)									8
									9

11.1.26.4. MODIFY POSITION

Dies ist der Befehl, um eine Position im File für TEACHING - Koordinaten am Computer zu ändern.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=68								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX=4								6
								7
POSITION NAME								8
								...
								23
DUMMY								24
HANDPOS								25
COORDINATES								26
								...
								67

POSITION NAME:

Name für die Position, 16 Byte im ASCII - CODE.

DUMMY:

Null. Nicht relevant.

HANDPOS:

Null.

COORDINATES:

Siehe Kapitel „Koordinaten“.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	
FUNCTION SUFFIX=4								6
								7
(ERRORCODE)								8
								9

11.1.26.5. DELETE POSITION

Dies ist der Befehl, um eine Position im File für TEACHING - Koordinaten am Computer zu löschen.

COMMAND:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=24								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0	0	1	
FUNCTION SUFFIX=5								6
								7
POSITION NAME								8
								...
								23

POSITION NAME:

Name für die Position, 16 Byte im ASCII - CODE.

RESPONSE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=8 (10 bei ERRORCODE)								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=72								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0	0	0/1	1	0	

FUNCTION SUFFIX=5	6
	7
(ERRORCODE)	8
	9

11.1.27. FUNCTION CODE 127: SPONTANEOUS MESSAGE

Dies ist eine Meldung der Roboter - Steuerung.

SPONTANEOUS MESSAGE:

7	6	5	4	3	2	1	0	BIT / BYTE
NOB=60								0
								1
DESTINATION ADDRESS								2
SOURCE ADDRESS								3
FUNCTION CODE=127								4
-	-	-	ORT	MLI	RS	T	T	5
0	0	0	0/1	0	0	0	3	
FUNCTION SUFFIX								6
								7
ERROR CODE / REGISTER NUMBER								8
								9
ERROR SUB NUMBER / REGISTER								10
								11
PROGRAM NUMBER								12
								13
INSTRUCTION NUMBER								14
								15

GOFLAG / ACTUAL TCP					16
LR	IR	PU	KEY	M O D E	17
COORDINATES					18
					...
					59

FUNCTION SUFFIX:

FUNCTION SUFFIX	Erklärung
0	SUCTRL = SUPERIOR CONTROL
1	EMERGENCY STOP
2	SEARCH STOP
3	PROGRAMMING UNIT STOP
4	KEY MODE CHANGE
5	PROGRAMMING UNIT DISCONNECT
6	ROBOT MODE CHANGE
7	SYSTEM ERROR
8	STARTUP
9	VISION SYSTEM INIT ERROR
10	VISION SYSTEM TEST ERROR
11	PROGRAM START
12	MODE TO OPERATE
13	MODE TO STAND BY
14	LIMIT EMERGENCY STOP
15	WORK STOP
16	SAFETY STOP
17	OPERATIONAL ERROR

ERROR CODE:

ERRORCODE. Nur relevant bei FUNCTION SUFFIX Sieben oder Siebzehn.

ERROR SUB NUMBER:

ERRORSUBCODE. Nur relevant bei FUNCTION SUFFIX Sieben oder Siebzehn.

REGISTER NUMBER:

Registernummer, 0-119. Nur relevant bei FUNCTION SUFFIX Null.

REGISTER:

Wert des Registers. Nur relevant bei FUNCTION SUFFIX Null.

PROGRAM NUMBER:

Aktives Programm.

INSTRUCTION NUMBER:

Aktive INSTRUCTION.

GOFLAG:

Null bedeutet STOP und Eins bedeutet CONTINUE. Nur relevant bei FUNCTION SUFFIX Null.

ACTUAL TCP:

Aktiver TCP. Nur relevant bei FUNCTION SUFFIX ungleich Null.

LR – LOCAL/REMOTE:

LR	Erklärung
0	LOCAL
1	REMOTE

IR – INTERRUPT:

IR	Erklärung
0	PERMITTED
1	NOT PERMITTED

PU – PROGRAMMING UNIT:

PU	Erklärung
0	CONNECTED
1	NOT CONNECTED

KEY (Schlüsselschalter):

KEY	Erklärung
0	AUTO
1	AUTO oder TEST 100%

MODE:

MODE	Erklärung
0	STAND BY
1	OPERATION
2	EXECUTION
3	EMERGENCY STOP

COORDINATES:

Siehe Kapitel „Koordinaten“.

11.2. Beispiel – Anwendung ORIDEMO, Listing

```
// -----  
// | ORI - Open Robot Interface, DEMO 1.0 |  
// -----  
// | This is the the ORI demo-application |  
// -----  
// | © 1998 by Stopper Markus, Angerer Bernhard, Panzirsch Robert |  
// -----  
  
unit uDemoForm;  
  
interface  
  
uses  
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
    StdCtrls, ExtCtrls, FileCtrl,  
    uDynConsoleForm,  
    uORIHelperClasses,  
    uEthernet,  
    uRS232,  
    uADLP10,  
    uARAP,  
    uORIAbstractClasses,  
    uRAL;  
  
////////////////////////////////////  
  
type  
    TFormDemo = class(TForm)  
        MemoORIDemo: TMemo;  
        ComboBoxValue: TComboBox;  
        ComboBoxOutput: TComboBox;  
        ButtonWriteOutput: TButton;  
        ButtonReturnHome: TButton;  
        RS232: TRS232;  
        Ethernet: TEthernet;
```



```
ADLP10: TADLP10;
RAL: TRAL;
Bevel2: TBevel;
Bevel3: TBevel;
ButtonSelectFolderTeaching: TButton;
EditSelectFileProgram: TEdit;
ButtonSelectFileProgram: TButton;
EditProgramNumber: TEdit;
ButtonSend: TButton;
Bevel4: TBevel;
ButtonReceive: TButton;
ButtonDiskLoad: TButton;
ButtonStartProgram: TButton;
ButtonStopProgram: TButton;
ButtonDeleteProgram: TButton;
ButtonReadProgramStatus: TButton;
ButtonWriteMode: TButton;
ButtonReadStatus: TButton;
ComboBoxMode: TComboBox;
ComboBoxTCP: TComboBox;
ButtonReadTCP: TButton;
ComboBoxLocation: TComboBox;
ButtonReadLocation: TButton;
ComboBoxSensor: TComboBox;
ButtonReadSensor: TButton;
ComboBoxFrame: TComboBox;
ButtonReadFrame: TButton;
ButtonWriteTCP: TButton;
ButtonWriteLocation: TButton;
ButtonWriteSensor: TButton;
ButtonWriteFrame: TButton;
Bevel7: TBevel;
ComboBoxRegister: TComboBox;
ButtonReadRegister: TButton;
ButtonWriteRegister: TButton;
EditConfig: TEdit;
ButtonReadConfig: TButton;
ButtonWriteConfig: TButton;
ComboBoxInput: TComboBox;
ButtonReadInput: TButton;
ButtonReadOutput: TButton;
```

```
EditX: TEdit;
EditY: TEdit;
EditZ: TEdit;
EditQ1: TEdit;
EditQ2: TEdit;
EditQ3: TEdit;
EditQ4: TEdit;
LabelX: TLabel;
LabelY: TLabel;
LabelZ: TLabel;
LabelQ1: TLabel;
LabelQ2: TLabel;
LabelQ3: TLabel;
LabelQ4: TLabel;
ButtonClearMemo: TButton;
ButtonMove: TButton;
Console1: TConsole;
OpenDialog: TOpenDialog;
EditBlockNumber: TEdit;
LabelProgram: TLabel;
LabelBlock: TLabel;
GroupBox1: TGroupBox;
RadioButtonProgram: TRadioButton;
RadioButton1: TRadioButton;
GroupBox2: TGroupBox;
RadioButtonWrist: TRadioButton;
RadioButtonTool: TRadioButton;
GroupBox3: TGroupBox;
RadioButtonRobot: TRadioButton;
RadioButtonRectangular: TRadioButton;
GroupBox4: TGroupBox;
RadioButtonAbsolut: TRadioButton;
RadioButtonRelative: TRadioButton;
EditV: TEdit;
LabelV: TLabel;
GroupBox5: TGroupBox;
RadioButtonBeginning: TRadioButton;
RadioButtonLastStop: TRadioButton;
CheckBoxTeaching: TCheckBox;
CheckBoxMessages: TCheckBox;
LabelBusy: TLabel;
```

```
Bevel1: TBevel;
Bevel5: TBevel;
ButtonSelectFileMessages: TButton;
Timer1: TTimer;
EditTeaching: TEdit;
ButtonActiveTCP: TButton;
EditHP: TEdit;
LabelHP: TLabel;
EditMessages: TEdit;
ARAP: TARAP;

procedure ButtonReadTCPClick(Sender: TObject);
procedure ButtonReadLocationClick(Sender: TObject);
procedure ButtonReadSensorClick(Sender: TObject);
procedure ButtonReadFrameClick(Sender: TObject);
procedure ButtonReadRegisterClick(Sender: TObject);
procedure ButtonReadConfigClick(Sender: TObject);
procedure ButtonReadInputClick(Sender: TObject);
procedure ButtonReadOutputClick(Sender: TObject);
procedure ButtonReadStatusClick(Sender: TObject);
procedure ButtonWriteTCPClick(Sender: TObject);
procedure ButtonWriteLocationClick(Sender: TObject);
procedure ButtonWriteSensorClick(Sender: TObject);
procedure ButtonWriteFrameClick(Sender: TObject);
procedure ButtonWriteRegisterClick(Sender: TObject);
procedure ButtonWriteConfigClick(Sender: TObject);
procedure ButtonWriteOutputClick(Sender: TObject);
procedure ButtonWriteModeClick(Sender: TObject);
procedure ButtonReturnHomeClick(Sender: TObject);
procedure ButtonSelectFileProgramClick(Sender: TObject);
procedure ButtonSendClick(Sender: TObject);
procedure ButtonReceiveClick(Sender: TObject);
procedure ButtonDiskLoadClick(Sender: TObject);
procedure ButtonStartProgramClick(Sender: TObject);
procedure ButtonStopProgramClick(Sender: TObject);
procedure ButtonDeleteProgramClick(Sender: TObject);
procedure ButtonReadProgramStatusClick(Sender: TObject);
procedure ButtonClearMemoClick(Sender: TObject);
procedure ButtonMoveClick(Sender: TObject);
procedure TeachingClick(Sender: TObject);
procedure MessagesClick(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
```

```
    procedure ButtonActiveTCPClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    FormDemo: TFormDemo;

const
    TCP12CONST=NULL10+#0+#0;           // for writing of a long TCP Value
    TCP6CONST=#0+#0+#0+#0+#0+#0;      // for writing of a short TCP Value
    LOCATIONCONST=NULL10+NULL10+NULL10+NULL10; // for writing of location
    SENSORCONST=NULL10;                // for writing of sensor
    FRAMECONST=NULL10+#0+#0+#0+#0;    // for writing of frame
    VALUECONST=#0+#0;                  // for writing of register value
    CONFIGCONST=#0;                    // for writing of config [...]

implementation

{$R *.DFM}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

procedure TFormDemo.ButtonActiveTCPClick(Sender: TObject);
begin
    if not RAL.ActiveTCP(StrToInt(ComboBoxTCP.Text))
    then MemoORIDemo.Lines.Add('Set Active TCP not possible !')
    else MemoORIDemo.Lines.Add('Set Active TCP successfully !');
end;

procedure TFormDemo.ButtonReadTCPClick(Sender: TObject);
var MemoString, TCP: string;
    i: integer;
begin
    MemoString:='';
    TCP:='';
    if not RAL.ReadTCP(StrToInt(ComboBoxTCP.Text), TCP)
    then MemoORIDemo.Lines.Add('Read TCP not possible !')
```

```
else begin
    for i:=1 to length(TCP) do
        MemoString:=MemoString+'#'+inttostr(ord(TCP[i]));
    MemoORIDemo.Lines.Add('TCP: '+MemoString);
    MemoORIDemo.Lines.Add('Read TCP successfully !')
end;
end;

procedure TFormDemo.ButtonReadLocationClick(Sender: TObject);
var MemoString, Location: string;
    i: integer;
begin
    MemoString:='';
    Location:='';
    if not RAL.ReadLocation(StrToInt(ComboBoxLocation.Text), Location)
    then MemoORIDemo.Lines.Add('Read Location not possible !')
    else begin
        for i:=1 to length(Location) do
            MemoString:=MemoString+'#'+inttostr(ord(Location[i]));
        MemoORIDemo.Lines.Add('Location: '+MemoString);
        MemoORIDemo.Lines.Add('Read Location successfully !')
    end;
end;

procedure TFormDemo.ButtonReadSensorClick(Sender: TObject);
var MemoString, Sensor: string;
    i: integer;
begin
    MemoString:='';
    Sensor:='';
    if not RAL.ReadSensor(StrToInt(ComboBoxSensor.Text), Sensor)
    then MemoORIDemo.Lines.Add('Read Sensor not possible !')
    else begin
        for i:=1 to length(Sensor) do
            MemoString:=MemoString+'#'+inttostr(ord(Sensor[i]));
        MemoORIDemo.Lines.Add('Sensor: '+MemoString);
        MemoORIDemo.Lines.Add('Read Sensor successfully !')
    end;
end;

procedure TFormDemo.ButtonReadFrameClick(Sender: TObject);
```

```
var MemoString, Frame: string;
    i: integer;
begin
    MemoString:='';
    Frame:='';
    if not RAL.ReadFrame(StrToInt(ComboBoxFrame.Text), Frame)
    then MemoORIDemo.Lines.Add('Read Frame not possible !')
    else begin
        for i:=1 to length(Frame) do
            MemoString:=MemoString+'#'+inttostr(ord(Frame[i]));
            MemoORIDemo.Lines.Add('Frame: '+MemoString);
            MemoORIDemo.Lines.Add('Read Frame successfully !')
        end;
    end;
end;

procedure TFormDemo.ButtonReadRegisterClick(Sender: TObject);
var MemoString, Value: string;
    i: integer;
begin
    MemoString:='';
    Value:='';
    if not RAL.ReadRegister(StrToInt(ComboBoxRegister.Text), Value)
    then MemoORIDemo.Lines.Add('Read Register not possible !')
    else begin
        for i:=1 to length(Value) do
            MemoString:=MemoString+'#'+inttostr(ord(Value[i]));
            MemoORIDemo.Lines.Add('Value: '+MemoString);
            MemoORIDemo.Lines.Add('Read Register successfully !')
        end;
    end;
end;

procedure TFormDemo.ButtonReadConfigClick(Sender: TObject);
var MemoString, Config: string;
    i: integer;
begin
    MemoString:='';
    Config:='';
    if not RAL.ReadConfig(StrToInt(EditConfig.Text), Config)
    then MemoORIDemo.Lines.Add('Read Config not possible !')
    else begin
        for i:=1 to length(Config) do
```

```
        MemoString:=MemoString+'#'+inttostr(ord(Config[i]));
MemoORIDemo.Lines.Add('Config: '+MemoString);
MemoORIDemo.Lines.Add('Read Config successfully !')
end;
end;

procedure TFormDemo.ButtonReadInputClick(Sender: TObject);
var Value: integer;
begin
    Value:=0;
    if not RAL.ReadInput(StrToInt(ComboBoxInput.Text),Value)
    then MemoORIDemo.Lines.Add('Read Input not possible !')
    else begin
        MemoORIDemo.Lines.Add('Input Value: #'+IntToStr(Value));
        MemoORIDemo.Lines.Add('Read Input successfully !')
    end;
end;

procedure TFormDemo.ButtonReadOutputClick(Sender: TObject);
var Value: integer;
begin
    Value:=0;
    if not RAL.ReadOutput(StrToInt(ComboBoxOutput.Text),Value)
    then MemoORIDemo.Lines.Add('Read Output not possible !')
    else begin
        MemoORIDemo.Lines.Add('Output Value: #'+IntToStr(Value));
        MemoORIDemo.Lines.Add('Read Output successfully !')
    end;
end;

procedure TFormDemo.ButtonReadStatusClick(Sender: TObject);
var MemoString, Status: string;
    i: integer;
begin
    MemoString:='';
    Status:='';
    if not RAL.ReadStatus(Status)
    then MemoORIDemo.Lines.Add('Read Status not possible !')
    else begin
        for i:=1 to length(Status) do
            MemoString:=MemoString+'#'+inttostr(ord(Status[i]));
```

```
        MemoORIDemo.Lines.Add('Status: '+MemoString);
        MemoORIDemo.Lines.Add('Read Status successfully !')
    end;
end;

procedure TFormDemo.ButtonWriteTCPClick(Sender: TObject);
var TCP: string;
begin
    if (StrToInt(ComboBoxTCP.Text)>=1)and(StrToInt(ComboBoxTCP.Text)<=19)
        then TCP:=TCP12CONST else TCP:=TCP6CONST;
    if not RAL.WriteTCP(StrToInt(ComboBoxTCP.Text), TCP)
        then MemoORIDemo.Lines.Add('Write TCP not possible !')
        else MemoORIDemo.Lines.Add('Write TCP successfully !');
end;

procedure TFormDemo.ButtonWriteLocationClick(Sender: TObject);
var Location: string;
begin
    Location:=LOCATIONCONST;
    if not RAL.WriteLocation(StrToInt(ComboBoxLocation.Text), Location)
        then MemoORIDemo.Lines.Add('Write Location not possible !')
        else MemoORIDemo.Lines.Add('Write Location successfully !');
end;

procedure TFormDemo.ButtonWriteSensorClick(Sender: TObject);
var Sensor: string;
begin
    Sensor:=SENSORCONST;
    if not RAL.WriteSensor(StrToInt(ComboBoxSensor.Text), Sensor)
        then MemoORIDemo.Lines.Add('Write Sensor not possible !')
        else MemoORIDemo.Lines.Add('Write Sensor successfully !');
end;

procedure TFormDemo.ButtonWriteFrameClick(Sender: TObject);
var Frame: string;
begin
    Frame:=FRAMECONST;
    if not RAL.WriteFrame(StrToInt(ComboBoxFrame.Text), Frame)
        then MemoORIDemo.Lines.Add('Write Frame not possible !')
        else MemoORIDemo.Lines.Add('Write Frame successfully !');
end;
```



```
procedure TFormDemo.ButtonWriteRegisterClick(Sender: TObject);
var Value: string;
begin
    Value:=VALUECONST;
    if not RAL.WriteRegister(StrToInt(ComboBoxRegister.Text), Value)
    then MemoORIDemo.Lines.Add('Write Register not possible !')
    else MemoORIDemo.Lines.Add('Write Register successfully !');
end;
```

```
procedure TFormDemo.ButtonWriteConfigClick(Sender: TObject);
var Config: string;
begin
    Config:=CONFIGCONST;
    if not RAL.WriteConfig(StrToInt(EditConfig.Text), Config)
    then MemoORIDemo.Lines.Add('Write Config not possible !')
    else MemoORIDemo.Lines.Add('Write Config successfully !');
end;
```

```
procedure TFormDemo.ButtonWriteOutputClick(Sender: TObject);
begin
    if not RAL.WriteOutput(StrToInt(ComboBoxOutput.Text),
        StrToInt(ComboBoxValue.Text))
    then MemoORIDemo.Lines.Add('Write Output not possible !')
    else MemoORIDemo.Lines.Add('Write Output successfully !');
end;
```

```
procedure TFormDemo.ButtonWriteModeClick(Sender: TObject);
begin
    if ComboBoxMode.ItemIndex=-1 then ComboboxMode.ItemIndex:=1;
    if not RAL.WriteMode(ComboBoxMode.ItemIndex)
    then MemoORIDemo.Lines.Add('Write Mode not possible !')
    else MemoORIDemo.Lines.Add('Write Mode successfully !');
end;
```

```
////////////////////////////////////
```

```
procedure TFormDemo.ButtonMoveClick(Sender: TObject);
var Vector: TRPoint;
    Orientation, Mode, Move, Velocity, HandPos: integer;
begin
```

```
if RadioButtonWrist.Checked=true then Orientation:=0 else Orientation:=1;
if RadioButtonAbsolut.Checked=true then Mode:=0 else Mode:=1;
if RadioButtonRectangular.Checked=true then Move:=0 else Move:=1;
Try
  Vector:=TRPoint.Create;
  Vector.X:=strtofloat(EditX.Text);
  Vector.Y:=strtofloat(EditY.Text);
  Vector.Z:=strtofloat(EditZ.Text);
  Vector.Q1:=strtofloat(EditQ1.Text);
  Vector.Q2:=strtofloat(EditQ2.Text);
  Vector.Q3:=strtofloat(EditQ3.Text);
  Vector.Q4:=strtofloat(EditQ4.Text);
  Velocity:=strtoint(EditV.Text);
  HandPos:=strtoint(EditHP.Text);
except
  ShowMessage('Please only Real Values !');
  EditX.Text:='950,0';
  EditY.Text:='0,0';
  EditZ.Text:='1585,0';
  EditQ1.Text:='1,0';
  EditQ2.Text:='0,0';
  EditQ3.Text:='0,0';
  EditQ4.Text:='0,0';
  EditV.Text:='200';
  EditHP.Text:='0';
  Vector.Destroy;
  EXIT;
end;
if not RAL.Move(Orientation, Mode, Move, Velocity, HandPos, Vector)
  then MemoORIDemo.Lines.Add('Move not possible !')
  else MemoORIDemo.Lines.Add('Move successfully !');
Vector.Destroy;
end;

procedure TFormDemo.ButtonReturnHomeClick(Sender: TObject);
begin
  if not RAL.ReturnHome
  then MemoORIDemo.Lines.Add('Return Home not possible !')
  else begin
    MemoORIDemo.Lines.Add('Return Home successfully !');
    EditX.Text:='950,0';
```

```
    EditY.Text:='0,0';
    EditZ.Text:='1585,0';
    EditQ1.Text:='1,0';
    EditQ2.Text:='0,0';
    EditQ3.Text:='0,0';
    EditQ4.Text:='0,0';
    EditV.Text:='200';
    EditHP.Text:='0';
end;
end;

////////////////////////////////////////////////////////////////

procedure TFormDemo.TeachingClick(Sender: TObject);
begin
    ARAP.Teaching:=CheckBoxTeaching.Checked;
    ARAP.TeachingFolder:=EditTeaching.Text;
    MemoORIDemo.Lines.Add('Change teaching successfully !');
end;

////////////////////////////////////////////////////////////////

procedure TFormDemo.MessagesClick(Sender: TObject);
begin
    ARAP.Messages:=CheckBoxMessages.Checked;
    ARAP.MessagesFile:=EditMessages.Text;
    MemoORIDemo.Lines.Add('Change messages successfully !');
end;

////////////////////////////////////////////////////////////////

procedure TFormDemo.ButtonSelectFileProgramClick(Sender: TObject);
begin
    if OpenFileDialog.Execute then
    begin
        if not FileExists(OpenDialog.FileName) then
        begin
            ShowMessage('This file does not exist');
            EXIT;
        end;
        EditSelectFileProgram.Text := OpenFileDialog.FileName;
    end;
end;
```

```
end;
end;

procedure TFormDemo.ButtonSendClick(Sender: TObject);
var i, Mode: integer;
    MemoString, ProgramData, Readchar: string;
    Readpchar: pchar;
    fs: TFileStream;
begin
    if RadioButtonProgram.Checked=true then Mode:=1 else Mode:=0;
    ProgramData:='';
    MemoString:='';
    try
        fs:=TFileStream.Create(EditSelectFileProgram.Text, fmOpenRead);
    except
        ShowMessage('Can not open file for reading !');
        EXIT;
    end;

    New(Readpchar);
    for i:=1 to fs.Size do
    begin
        fs.Read(Readpchar^, 1);           // read one byte
        Readchar:=Readpchar;             // convert to string
        if Readchar='' then Readchar:=#0; // byte=#0
        Readchar:=copy(Readchar,1,1);     // 1 byte
        ProgramData:=ProgramData+Readchar; // collect
    end;
    Dispose(Readpchar);
    fs.Destroy;

    for i:=1 to length(ProgramData) do
        MemoString:=MemoString+'#'+inttostr(ord(ProgramData[i]));
    MemoORIDemo.Lines.Add('ProgramData: '+MemoString);

    if not RAL.SendProgram(Mode, StrToInt(EditProgramNumber.Text),
        StrToInt(EditBlockNumber.Text), Programdata)
    then MemoORIDemo.Lines.Add('Send Program not possible !')
    else MemoORIDemo.Lines.Add('Send Program successfully !');
end;
```

```
procedure TFormDemo.ButtonReceiveClick(Sender: TObject);
var i, Mode: integer;
    MemoString, ProgramData: string;
    fs: TFileStream;
begin
    if RadioButtonProgram.Checked=true then Mode:=1 else Mode:=0;
    ProgramData:='';
    MemoString:='';
    if not RAL.ReceiveProgram(Mode, StrToInt(EditProgramNumber.Text),
                               StrToInt(EditBlockNumber.Text), Programdata)
    then MemoORIDemo.Lines.Add('Receive Program not possible !')
    else begin
        for i:=1 to length(ProgramData) do
            MemoString:=MemoString+'#'+inttostr(ord(ProgramData[i]));
        MemoORIDemo.Lines.Add('ProgramData: '+MemoString);
        try
            fs:=TFileStream.Create(EditSelectFileProgram.Text, fmOpenWrite);
        except
            try
                fs:=TFileStream.Create(EditSelectFileProgram.Text, fmCreate);
            except
                ShowMessage('Can not open file for writing !');
                EXIT;
            end;
        end;
    end;
    try
        fs.Write(PChar(ProgramData)^, Length(ProgramData)); // write all bytes
    except
        ShowMessage('Can not write to file !');
        MemoORIDemo.Lines.Add('Receive Program not possible !');
        fs.Destroy;
        EXIT;
    end;
    fs.Destroy;
    MemoORIDemo.Lines.Add('Receive Program successfully !');
end;

end;

procedure TFormDemo.ButtonDiskLoadClick(Sender: TObject);
var Mode: integer;
begin
```

```
if RadioButtonProgram.Checked=true then Mode:=1 else Mode:=0;
if not RAL.DiskLoadProgram(Mode, StrToInt(EditProgramNumber.Text),
    StrToInt(EditBlockNumber.Text))
    then MemoORIDemo.Lines.Add('Disk Load Program not possible !')
    else MemoORIDemo.Lines.Add('Disk Load Program successfully !');
end;

procedure TFormDemo.ButtonStartProgramClick(Sender: TObject);
var FirstInstruction: integer;
begin
    if RadioButtonBeginning.Checked=true then FirstInstruction:=0
        else FirstInstruction:=1;
    if not RAL.StartProgram(FirstInstruction, StrToInt(EditProgramNumber.Text))
        then MemoORIDemo.Lines.Add('Start Program not possible !')
        else MemoORIDemo.Lines.Add('Start Program successfully !');
end;

procedure TFormDemo.ButtonStopProgramClick(Sender: TObject);
begin
    if not RAL.StopProgram
        then MemoORIDemo.Lines.Add('Stop Program not possible !')
        else MemoORIDemo.Lines.Add('Stop Program successfully !');
end;

procedure TFormDemo.ButtonDeleteProgramClick(Sender: TObject);
begin
    if not RAL.DeleteProgram(StrToInt(EditProgramNumber.Text))
        then MemoORIDemo.Lines.Add('Delete Program not possible !')
        else MemoORIDemo.Lines.Add('Delete Program successfully !');
end;

procedure TFormDemo.ButtonReadProgramStatusClick(Sender: TObject);
var MemoString, ProgramStatus: string;
    i: integer;
begin
    if not RAL.ReadProgramStatus(ProgramStatus)
        then MemoORIDemo.Lines.Add('Program Status not possible !')
        else begin
            for i:=1 to length(ProgramStatus) do
                MemoString:=MemoString+'#'+inttostr(ord(ProgramStatus[i]));
            MemoORIDemo.Lines.Add('Program Status: '+MemoString);
        end;
end;
```

```
        MemoORIDemo.Lines.Add('Program Status successfully !');
    end;
end;
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
procedure TFormDemo.ButtonClearMemoClick(Sender: TObject);
begin
    MemoORIDemo.Lines.Clear;
end;
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
procedure TFormDemo.Timer1Timer(Sender: TObject);
// BusyFlag
begin
    if RAL.Busy then LabelBusy.Enabled:=true else LabelBusy.Enabled:=false;
end;

end.
```

11.3. Literaturverzeichnis

- [STOPPER] Dipl.-Ing. Stopper Markus: AN ARCHITECTURE FOR INDUSTRIAL ROBOT REMOTE PROGRAMMING USING A DISTRIBUTED COMPUTING ENVIRONMENT, DANUBE ADRIA ASSOCIATION FOR AUTOMATION & MANUFACTURING, VOLUME 2/2, Seite 453-455, Wien, Oktober 1998
- [ADLP10] THE ABB DATA LINK PROTOCOL, ABB ROBOTICS PRODUCTS, Schweden, März 1993
- [ARAP] THE ABB ROBOT APPLICATION PROTOCOL S3, ABB ROBOTICS PRODUCTS, Schweden, März 1993
- [IRB2000] ABB IRB 2000 / S3 INSTALLATION MANUAL, ABB ROBOTICS PRODUCTS, Schweden, August 1991
- [PRG3] Handbuch Programmier - Grundkurs PRG 3, ABB Roboter GmbH, Friedberg, Oktober 1988
- [DELPHI21] Kent Reisdorph: BORLAND DELPHI 4 IN 21 DAYS, SAMS TEACH YOURSELF, BORLAND PRESS, ISBN 0-672-31286-7, Juli 1998
- [CALVERT] Charles Calvert: DELPHI 4 UNLEASHED, BORLAND PRESS/SAMS, ISBN 0-672-31285-9, September 1998
- [DGUIDE] Xavier Pacheco and Steve Teixeira: DELPHI 4 DEVELOPER'S GUIDE, BORLAND PRESS/SAMS, ISBN 0-672-31284-0, 1998

11.4. ASCII - CODE Tabelle

Für die Kommunikation werden folgende ASCII - CODES als Steuerzeichen verwendet:

Dezimal	Befehl	Erklärung
2	STX (even)	Gerades Textanfangszeichen
3	ETX	Textendezeichen
4	EOT	Ende der Übertragung
5	ENQ	Aufforderung zur Datenübertragung
6	ACK	Positive Rückmeldung
10	LF	Zeilenvorschub
13	CR	Wagenrücklauf
14	WACK	Positive Rückmeldung und Warten
15	RVI	Unterbrechung zur Richtungsumschaltung
16	DLE	Datenübertragungsumschaltung
17	XON	Gerätsteuerzeichen 1
19	XOFF	Gerätsteuerzeichen 3
21	NAK	Negative Rückmeldung
130	STX (odd)	Ungerades Textanfangszeichen