Perfection in Automation
www.br-automation.com

BrbLibVc2d V1.02
Dokumentation

B&R übernimmt keine Haftung für Folgen, die durch die Implementierung sowie die Benutzung dieser Software entstehen!

I

Inhaltsverzeichnis

1 Allgemeines

Die Bibliothek „BrbLibVc2d“ enthält Funktionen zum Darstellen einer dynamischen 2d-Zeichnung in einer Vc4-Visualisierung.

Diese Bibliothek ist keine offizielle B&R-Software. Es besteht kein Anspruch auf Support, Wartung oder Fehlerbehebung. Die Benutzung geschieht auf eigene Gefahr.

Die Bibliothek unterliegt der MIT-Lizenz (siehe ‚License.txt‘), welche zwar unbeschränkte Nutzung auf eigene Gefahr gewährt, jedoch alle Haftungsansprüche ausschließt.

1.1 Hinweise zum Compiler

Das Entwicklungs- und Demo-Projekt ist auf den Compiler V6.3.0 gesetzt, mit dem das Projekt und damit auch die Bibliothek fehler- und warnungslos kompiliert werden können.
Die Bibliothek ist aber auch unter älteren Compiler-Versionen einsetzbar.

1.2 Abhängigkeiten

Es besteht eine Abhängigkeit von folgenden Bibliotheken:

-BrbLib V5.03
-BrbLibVc4 V5.03

1.3 Hinweise zu StructuredText und anderen IEC-Sprachen

Die Bibliothek ist in ANSI-C geschrieben, kann aber auch in StructuredText und allen anderen IEC-Sprachen verwendet werden.

Einschränkung:
Beim Zeichnen eines 2d-Modells sind optional über sogenannte Funktionszeiger benutzerdefinierte Erweiterungen implementiert.
Da die IEC-Sprachen keine Funktionszeiger unterstützen, sind diese Erweiterungen nur in ANSI-C nutzbar. Die entsprechenden Eingänge der Funktionen für die Funktionszeiger müssen in IEC-Sprachen auf 0 gesetzt werden. Ansonsten können auch diese Funktionen ohne Probleme verwendet werden.

1.4 Geprüft mit ClangTidy

Das gesamte Entwicklungs- und Demo-Projekt wurde mit dem Code-Analyse-Tool ClangTidy geprüft (Details siehe Dokumentation der Basis-Bibliothek „BrbLib“).

1.5 Quellcode und Binär-Variante der Bibliothek

Im Release der Bibliothek ist sowohl die Quellcode- als auch die Binär-Variante der Bibliothek enthalten. Beide Varianten sind komplett identisch.
Welche Variante der Anwender in sein Projekt einfügt, sollte von diesen Punkten abhängig gemacht werden:

1.5.1 Quellcode-Variante

Sie enthält den kompletten Quellcode aller Funktionen in ANSI-C. Somit kann der Anwender diesen studieren und unter Umständen eine ähnliche/abgewandelte Funktion sehr leicht in einer eigenen Bibliothek implementieren. Auch das Online-Debuggen durch Breakpoints ist möglich.
Beim Rebuild wird allerdings auch diese Bibliothek nochmals kompiliert. Dies kann je nach verwendetem Rechner einige Zeit in Anspruch nehmen.
Hinweis: Von der Änderung der Funktionen in der ausgelieferten Bibliothek wird abgeraten, da dann ein Umstieg auf eine neuere Version schwierig bis unmöglich wird.

1.5.2 Binär-Variante

Sie enthält nur vorkompilierte Module der Bibliothek. Es ist also kein Quellcode enthalten. Der Vorteil besteht darin, dass die Bibliothek auch bei einem Rebuild nicht mehr kompiliert werden muss. Dies bedeutet unter Umständen einen großen Zeitvorteil.

1.5.2.1 Hinweise zu verschiedenen Prozessoren (Intel + ARM)

Die im Release enthaltene Binär-Variante ist für Zielsysteme mit Intel-Prozessor (SG4) exportiert und nur dort lauffähig.
Soll eine Binär-Variante für ein Zielsystem mit ARM-Prozessor (SGC) eingesetzt werden, so muss eine eigens exportierte Binär-Version verwendet werden. Das Exportieren einer Code-Bibliothek als Binär-Version ist in der AS-Hilfe unter der GUID `d750bdd3-0aad-4486-8c0d-4eb43372b325` beschrieben.
Achtung: Bei einer für ARM-Prozessoren exportierten Binär-Bibliothek kann nicht in den ArSim-Modus geschalten werden, da ArSim wiederum die Intel-Version benötigt. Soll ArSim verfügbar sein, muss die Quellcode-Variante der Bibliothek eingefügt werden, denn nur dann kann sie je nach Modus kompiliert werden.

1.6 Neueste Versionen auf GitHub

GitHub ist eine öffentliche Plattform für kostenlose Software. Der Download ist ohne Anmeldung möglich. Darauf sind verschiedene Pakete des Autors kostenlos erhältlich. Sie unterliegen alle der MIT-Lizenz (siehe oben).

Die Bibliothek BrbLibVc2d und dessen unterlagerten Bibliothek BrbLib und BrbLibVc4 sind als eigenes Release-Paket erhältlich. Es enthält die Bibliotheken u.a. in Sourcecode- und Binär-Version:
https://github.com/br-automation-com/BrbLibs-lib-src/releases

Auch erhältlich ist das Windows-Tool ‚RnCommTest' zum Testen von Kommunikationen. Es enthält u.a. folgende Module:
-Serielle Kommunikation (RS232/485)
-Tcp-Client, Tcp-Server
-Udp
-ModbusTcp-Master, ModbusTcp-Client
-OpcUa-Client, OpcUa-Server, OpcUa-Subscriber
Es ist unter diesem Link erhältlich:
https://github.com/br-automation-com/RnCommTest-Windows/releases

Außerdem gibt es ein Beispiel-Projekt für OpcUa inklusive der Bibliothek BrbLibUa:
https://github.com/br-automation-com/OpcUaSamples-sample-AS/releases

2 Revisionsgeschichte

Hier ist hier nur die letzte Version erwähnt. Die gesamte Revisionsgeschichte ist in die Datei „BrbLibVc2d Revisionsgeschichte“ ausgelagert.

2.1 BrbLibVc2d V1.02 – 2024-01-10

2.1.1 Hinweise zu Prozessoren bei Binär-Version

In diese Hilfe wurden die Hinweise zu verschiedenen Prozessoren (Intel + ARM) bei Verwendung der Binär-Version aufgenommen.

2.1.2 Hinweise zum Compiler

In diese Hilfe wurden die Hinweise zum Compiler aufgenommen.

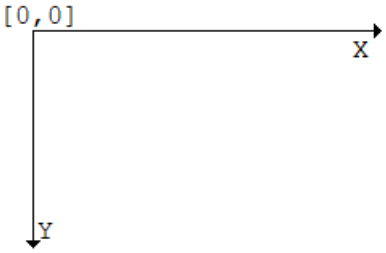
2.1.3 Code-Prüfung mit ClangTidy

Die Prüfung mit ClangTidy des Entwicklungs- und Demo-Projekts (siehe Allgemeines/Geprüft mit ClangTidy) wurde mit einer neuen Version durchgeführt (die vorige Version prüfte manche Tasks nicht komplett). Die dadurch erkannten Code-Stellen wurden überprüft und gegebenenfalls optimiert. Die Bibliothek wurde dabei nicht geändert.

3 Allgemeines zu 2d-Grafik

3.1 Koordinaten-System

Da in einer DrawBox (Vc4-Control zum programmgesteuerten Zeichnen, siehe AS-Hilfe) gezeichnet wird, wurde auch das Koordinaten-System der DrawBox übernommen.
Dabei ist die linke, obere Ecke der Punkt [0,0]. Die X-Achse wird nach rechts größer, die Y-Achse nach unten:



Normalerweise drehen Geometrie-Formeln zur 2d-Berechnung bei positiven Winkeln immer gegen den Uhrzeigersinn. Um die Anwendung intuitiver zu machen, drehen sie hier stets **im** Uhrzeigersinn.

4 Pakete

4.1 Einfacher Punkt

Elemente (siehe unten) sind durch mehrere Punkte definiert. Es gibt daher eine Struktur, mit der ein einfacher Punkt dargestellt wird:

BrbVc2dPoint_TYP		
rX	REAL	X Koordinate
rY	REAL	Y-Koordinate

4.1.1 BrbVc2dSetSimplePoint

```
unsigned short BrbVc2dSetSimplePoint(struct BrbVc2dPoint_TYP* pPoint, float rX, float rY)
```

Argumente:

`struct BrbVc2dPoint_TYP* pPoint`
Zeiger auf den einfachen Punkt

`REAL rX`
X-Koordinate

`REAL rY`
Y-Koordinate

Rückgabe:

`UINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion besetzt einen einfachen Punkt mit den übergegebenen Werten. Sie wird normalerweise nur intern und nicht applikativ verwendet.

4.2 Elemente

Elemente sind primitive 2d-Formen, aus denen Modelle zusammengestellt werden können.
Jedes Element kann dann zur Laufzeit dynamisiert werden (z.B. verschoben oder gedreht). Daraus entsteht dann das komplett animierte Modell.
Folgende Element-Formen sind möglich: Punkt, Linie, Dreieck, Viereck, Kreis und Ellipse.

4.2.1 Element-Struktur

Jede Element-Form wird durch dieselbe Struktur definiert:

BrbVc2dElement_TYP		
eElementType	BrbVc2dElements_ENUM	Eingang: Typ des Elements
bDraw	BOOL	Eingang: 1=Zeichnen
Points	BrbVc2dPoint_TYP[0..3]	Eingang: Punkte des Elements
nBorderColor	USINT[0..3]	Eingang: Farbe einer Rand-Linie
nFillColor	USINT	Eingang: Füll-Farbe

`eElementType` gibt die Grundform des Elements an (Detail-Beschreibung siehe weiter unten):

BrbVc2dElements_ENUM	
eBRBVC2D_ELEMENT_POINT	0=Punkt
eBRBVC2D_ELEMENT_LINE	1=Linie
eBRBVC2D_ELEMENT_TRIANGLE	2=Dreieck
eBRBVC2D_ELEMENT_TETRAGON	3=Viereck
eBRBVC2D_ELEMENT_CIRCLE	4=Kreis
eBRBVC2D_ELEMENT_ELLIPSE	5=Ellipse

`bDraw` gibt an, ob das Element tatsächlich gezeichnet werden soll oder nicht. Damit können auch Elemente vorhanden sein, welche nur zur Dynamisierung verwendet werden (z.B. können die Punkte eines nicht gezeichneten Dreiecks als Drehpunkt für andere Elemente benutzt werden).

`Points` gibt die Definition des Elements an. Je nach Form werden dabei verschieden viele dieser insgesamt 4 Punkte verwendet (siehe unten).

`nBorderColor` gibt die Farbe der Rahmenlinien an. Je nach Form werden dabei verschieden viele dieser insgesamt 4 Farben verwendet. Angegeben wird hier der Index aus der Vc4-Farb-Palette.
Der Index 255 entspricht einer Sonder-Funktion. Linien mit diesem Index werden nicht gezeichnet, sind also transparent. Damit können nahtlose Übergänge zwischen zwei Elementen realisiert werden.

`nFillColor` gibt die Füllfarbe für Elemente an, die eine Fläche besitzen. Angegeben wird hier der Index aus der Vc4-Farb-Palette.
Auch hier entspricht der Index 255 einer transparenten Fläche.

Da jede Form mit derselben Struktur abgebildet wird, kann mit einem Array ein komplettes Modell erzeugt werden (siehe weiter unten).

4.2.2 Punkt

4.2.2.1 Benutzte Items der Element-Struktur

Item	Datentyp	Beschreibung
eElementType	BrbVc2dElements_ENUM	= eBRBVC2D_ELEMENT_POINT
bDraw	BOOL	1 = Element wird gezeichnet
Point[0]	BrbVc2dPoint_TYP	Die Koordinaten des Punkts
nFillColor	USINT	Die Farbe des Punkts 0..254 = Index der Vc4-Farb-Palette 255 = Transparent

4.2.2.2 BrbVc2dSetElementPoint

```
unsigned short BrbVc2dSetElementPoint(struct BrbVc2dElement_TYP* pElement, float rX, float rY,
unsigned char nFillColor, plcbit bDraw)
```

Argumente:

`struct BrbVc2dElement_TYP* pElement`
Zeiger auf das Element

`REAL rX`
X-Koordinate

`REAL rY`

Y-Koordinate
`USINT rFillColor`
Farbe
`BOOL bDraw`
1=Element wird gezeichnet

Rückgabe:
`UINT`
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`

Beschreibung:
Diese Funktion besetzt ein Punkt-Element mit den übergegebenen Werten.

4.2.3 Linie

4.2.3.1 Benutzte Items der Element-Struktur

Item	Datentyp	Beschreibung
eElementType	BrbVc2dElements_ENUM	= eBRBVC2D_ELEMENT_LINE
bDraw	BOOL	1 = Element wird gezeichnet
Point[0]	BrbVc2dPoint_TYP	Die Koordinaten des Start-Punkts
Point[1]	BrbVc2dPoint_TYP	Die Koordinaten des End-Punkts
nFillColor	USINT	Die Farbe der Linie 0..254 = Index der Vc4-Farb-Palette 255 = Transparent

4.2.3.2 BrbVc2dSetElementLine

```
unsigned short BrbVc2dSetElementLine(struct BrbVc2dElement_TYP* pElement, float rX0, float rY0,
float rX1, float rY1, unsigned char nFillColor, plcbit bDraw)
```

Argumente:
`struct BrbVc2dElement_TYP* pElement`
Zeiger auf das Element
`REAL rX0`
X-Koordinate des Start-Punkts
`REAL rY0`
Y-Koordinate des Start-Punkts
`REAL rX1`
X-Koordinate des End-Punkts
`REAL rY1`
Y-Koordinate des End-Punkts
`USINT rFillColor`
Farbe
`BOOL bDraw`
1=Element wird gezeichnet

Rückgabe:
`UINT`
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`

Beschreibung:
Diese Funktion besetzt ein Linien-Element mit den übergegebenen Werten.

4.2.4 Dreieck

4.2.4.1 Benutzte Items der Element-Struktur

Item	Datentyp	Beschreibung
eElementType	BrbVc2dElements_ENUM	= eBRBVC2D_ELEMENT_TRIANGLE
bDraw	BOOL	1 = Element wird gezeichnet
Point[0]	BrbVc2dPoint_TYP	Die Koordinaten des 1.Punkts
Point[1]	BrbVc2dPoint_TYP	Die Koordinaten des 2.Punkts
Point[2]	BrbVc2dPoint_TYP	Die Koordinaten des 3.Punkts

nBorderColor[0]	USINT	Die Farbe der Randlinie zwischen Punkt 1 und 2 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nBorderColor[1]	USINT	Die Farbe der Randlinie zwischen Punkt 2 und 3 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nBorderColor[2]	USINT	Die Farbe der Randlinie zwischen Punkt 3 und 1 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nFillColor	USINT	Die Füll-Farbe 0..254 = Index der Vc4-Farb-Palette 255 = Transparent

4.2.4.2 BrbVc2dSetElementTriangle

```
unsigned short BrbVc2dSetElementTriangle(struct BrbVc2dElement_TYP* pElement, float rX0, float
rY0, unsigned char nBorderColor0, float rX1, float rY1, unsigned char nBorderColor1, float rX2,
float rY2, unsigned char nBorderColor2, unsigned char nFillColor, plcbit bDraw)
```

Argumente:

```
struct BrbVc2dElement_TYP* pElement
```
Zeiger auf das Element

```
REAL rX0
```
X-Koordinate des 1.Punkts

```
REAL rY0
```
Y-Koordinate des 1.Punkts

```
USINT nBorderColor0
```
Farbe der Randlinie zwischen Punkt 1 und 2

```
REAL rX1
```
X-Koordinate des 2.Punkts

```
REAL rY1
```
Y-Koordinate des 2.Punkts

```
USINT nBorderColor1
```
Farbe der Randlinie zwischen Punkt 2 und 3

```
REAL rX2
```
X-Koordinate des 3.Punkts

```
REAL rY2
```
Y-Koordinate des 3.Punkts

```
USINT nBorderColor2
```
Farbe der Randlinie zwischen Punkt 3 und 1

```
USINT rFillColor
```
Füll-Farbe

```
BOOL bDraw
```
1=Element wird gezeichnet

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion besetzt ein Dreieck-Element mit den übergegebenen Werten.

4.2.5 Viereck (Tetragon)

Hinweis: Ein Viereck muss nicht zwingend rechtwinklig sein. Auch die Seiten können unterschiedlich lang sein.

4.2.5.1 Benutzte Items der Element-Struktur

Item	Datentyp	Beschreibung
eElementType	BrbVc2dElements_ENUM	= eBRBVC2D_ELEMENT_TETRAGON
bDraw	BOOL	1 = Element wird gezeichnet
Point[0]	BrbVc2dPoint_TYP	Die Koordinaten des 1.Punkts
Point[1]	BrbVc2dPoint_TYP	Die Koordinaten des 2.Punkts
Point[2]	BrbVc2dPoint_TYP	Die Koordinaten des 3.Punkts
Point[3]	BrbVc2dPoint_TYP	Die Koordinaten des 4.Punkts
nBorderColor[0]	USINT	Die Farbe der Randlinie zwischen Punkt 1 und 2 0..254 = Index der Vc4-Farb-Palette

		255 = Transparent
nBorderColor[1]	USINT	Die Farbe der Randlinie zwischen Punkt 2 und 3 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nBorderColor[2]	USINT	Die Farbe der Randlinie zwischen Punkt 3 und 4 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nBorderColor[3]	USINT	Die Farbe der Randlinie zwischen Punkt 4 und 1 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nFillColor	USINT	Die Füll-Farbe 0..254 = Index der Vc4-Farb-Palette 255 = Transparent

4.2.5.2 BrbVc2dSetElementTetragon

```
unsigned short BrbVc2dSetElementTetragon(struct BrbVc2dElement_TYP* pElement, float rX0, float
rY0, unsigned char nBorderColor0, float rX1, float rY1, unsigned char nBorderColor1, float rX2,
float rY2, unsigned char nBorderColor2, float rX3, float rY3, unsigned char nBorderColor3, un-
signed char nFillColor, plcbit bDraw)
```

Argumente:

```
struct BrbVc2dElement_TYP* pElement
```
Zeiger auf das Element

```
REAL rX0
```
X-Koordinate des 1.Punkts

```
REAL rY0
```
Y-Koordinate des 1.Punkts

```
USINT nBorderColor0
```
Farbe der Randlinie zwischen Punkt 1 und 2

```
REAL rX1
```
X-Koordinate des 2.Punkts

```
REAL rY1
```
Y-Koordinate des 2.Punkts

```
USINT nBorderColor1
```
Farbe der Randlinie zwischen Punkt 2 und 3

```
REAL rX2
```
X-Koordinate des 3.Punkts

```
REAL rY2
```
Y-Koordinate des 3.Punkts

```
USINT nBorderColor2
```
Farbe der Randlinie zwischen Punkt 3 und 4

```
REAL rX3
```
X-Koordinate des 4.Punkts

```
REAL rY3
```
Y-Koordinate des 4.Punkts

```
USINT nBorderColor3
```
Farbe der Randlinie zwischen Punkt 4 und 1

```
USINT rFillColor
```
Füll-Farbe

```
BOOL bDraw
```
1=Element wird gezeichnet

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion besetzt ein Viereck-Element mit den übergegebenen Werten.

4.2.5.3 BrbVc2dSetElementTetragon2

```
unsigned short BrbVc2dSetElementTetragon2(struct BrbVc2dElement_TYP* pElement, float rX, float
rY, float rWidth, float rHeight, unsigned char nBorderColor0, unsigned char nBorderColor1, un-
signed char nBorderColor2, unsigned char nBorderColor3, unsigned char nFillColor, plcbit bDraw)
```

Argumente:

```
struct BrbVc2dElement_TYP* pElement
```
Zeiger auf das Element

```
REAL rX
```

X-Koordinate des Start-Punkts
```
REAL rY
```
Y-Koordinate des Start-Punkts
```
REAL rWidth
```
Breite des Rechtecks
```
REAL rHeight
```
Höhe des Rechtecks
```
USINT nBorderColor0
```
Farbe der Randlinie zwischen Punkt 1 und 2
```
USINT nBorderColor1
```
Farbe der Randlinie zwischen Punkt 2 und 3
```
USINT nBorderColor2
```
Farbe der Randlinie zwischen Punkt 3 und 4
```
USINT nBorderColor3
```
Farbe der Randlinie zwischen Punkt 4 und 1
```
USINT rFillColor
```
Füll-Farbe
```
BOOL bDraw
```
1=Element wird gezeichnet

Rückgabe:
```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion besetzt ein Viereck-Element mit den übergegebenen Werten. Dabei kann die Breite und Höhe angegeben werden, so dass immer ein gerades, gleichseitiges Rechteck entsteht.

4.2.6 Kreis

Hinweis: Ein Kreis benötigt zum Zeichnen weniger Performance als eine Ellipse.

4.2.6.1 Benutzte Items der Element-Struktur

Item	Datentyp	Beschreibung
eElementType	BrbVc2dElements_ENUM	= eBRBVC2D_ELEMENT_CIRCLE
bDraw	BOOL	1 = Element wird gezeichnet
Point[0]	BrbVc2dPoint_TYP	Die Koordinaten des Mittelpunkts
Point[1].rX	REAL	Der Radius des Kreises
nBorderColor[0]	USINT	Die Farbe der Randlinie 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nFillColor	USINT	Die Füll-Farbe 0..254 = Index der Vc4-Farb-Palette 255 = Transparent

4.2.6.2 BrbVc2dSetElementCircle

```
unsigned short BrbVc2dSetElementCircle(struct BrbVc2dElement_TYP* pElement, float rX, float rY,
float rRadius, unsigned char nBorderColor, unsigned char nFillColor, plcbit bDraw)
```

Argumente:
```
struct BrbVc2dElement_TYP* pElement
```
Zeiger auf das Element
```
REAL rX
```
X-Koordinate des Mittelpunkts
```
REAL rY
```
Y-Koordinate des Mittelpunkts
```
REAL rRadius
```
Radius des Kreises
```
USINT nBorderColor
```
Farbe der Randlinie
```
USINT rFillColor
```
Füll-Farbe
```
BOOL bDraw
```
1=Element wird gezeichnet

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion besetzt ein Kreis-Element mit den übergegebenen Werten.

4.2.7 Ellipse

Hinweis: Eine Ellipse benötigt zum Zeichnen mehr Performance als ein Kreis.

4.2.7.1 Benutzte Items der Element-Struktur

Item	Datentyp	Beschreibung
eElementType	BrbVc2dElements_ENUM	= eBRBVC2D_ELEMENT_ELLIPSE
bDraw	BOOL	1 = Element wird gezeichnet
Point[0]	BrbVc2dPoint_TYP	Die Koordinaten des Mittelpunkts
Point[1].rX	REAL	Der Radius der Ellipse auf der X-Achse
Point[1].rY	REAL	Der Radius der Ellipse auf der Y-Achse
Point[2].rX	REAL	Dreh-Winkel um den Mittelpunkt in [°]
nBorderColor[0]	USINT	Die Farbe der Randlinie 0..254 = Index der Vc4-Farb-Palette 255 = Transparent
nFillColor	USINT	Die Füll-Farbe 0..254 = Index der Vc4-Farb-Palette 255 = Transparent

4.2.7.2 BrbVc2dSetElementEllipse

```
unsigned short BrbVc2dSetElementEllipse(struct BrbVc2dElement_TYP* pElement, float rX, float rY,
float rRadiusX, float rRadiusY, float rAngle, unsigned char nBorderColor, unsigned char nFillCol-
or, plcbit bDraw)
```

Argumente:

```
struct BrbVc2dElement_TYP* pElement
```
Zeiger auf das Element

```
REAL rX
```
X-Koordinate des Mittelpunkts

```
REAL rY
```
Y-Koordinate des Mittelpunkts

```
REAL rRadiusX
```
Radius der Ellipse auf der X-Achse

```
REAL rRadiusY
```
Radius der Ellipse auf der Y-Achse

```
REAL rAngle
```
Dreh-Winkel um den Mittelpunkt in [°]

```
USINT nBorderColor
```
Farbe der Randlinie

```
USINT rFillColor
```
Füll-Farbe

```
BOOL bDraw
```
1=Element wird gezeichnet

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion besetzt ein Ellipsen-Element mit den übergegebenen Werten.

4.2.8 Hilfsfunktionen für Elemente

4.2.8.1 BrbVc2dDrawElement

```
unsigned short BrbVc2dDrawElement(struct BrbVc2dElement_TYP* pElement, struct BrbVc4General_TYP*
pGeneral)
```

Argumente:

```
struct BrbVc2dElement_TYP* pElement
```
Zeiger auf das Element
```
struct BrbVc4General_TYP* pGeneral
```
Zeiger auf die Instanz von „BrbVc4General_TYP“

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion zeichnet ein Element in die aktuell referenzierte Drawbox.
Sie wird normalerweise nicht applikativ benötigt, sondern nur intern verwendet.

4.3 Modell

Ein Modell wird durch ein Array von Elementen realisiert. Durch die Dynamisierung (siehe unten) einiger oder aller Elemente entsteht der Eindruck einer Bewegung.

4.3.1 Struktur

BrbVc2dModel_TYP		
bEnable	BOOL	Eingang: 1=Zeichnen
Cfg	BrbVc2dModelCfg_TYP	Konfiguration des Modells
Intern	BrbVc2dModelIntern_TYP	Interne Variablen des Modells

Das Modell wird nur gezeichnet, wenn der Eingang „bEnable“ auf 1 ist.
Das Modell muss nicht jeden Task-Zyklus neu gezeichnet werden. Siehe dazu „Allgemeine Hinweise zur Dynamisierung eines Modells“ weiter unten.

4.3.1.1 Konfiguration

BrbVc2dModelCfg_TYP		Konfiguration des Modells
Drawbox	BrbVc4Drawbox_TYP	Angaben zur Drawbox
Padding	BrbVc2dModelPadding_TYP	Einrückung des Models
rScaling	REAL	Eingang: Skalierung des Models
nElementIndexMax	DINT	Eingang: Maximaler Index des Element-Arrays
& pElements	BrbVc2dElement_TYP	Eingang: Zeiger auf das Element-Array
Callbacks	BrbVc2dModelCfgCallbacks_TYP	Konfiguration der Aufrufe
pTag	UDINT	Eingang: Zeiger auf Benutzer-Daten

Die Konfiguration ist der Übersichtlichkeit wegen in verschiedene Unter-Strukturen aufgeteilt.

4.3.1.1.1 Allgemeines

BrbVc4Drawbox_TYP		
nLeft	UDINT	Eingang: Linke Koordinate
nTop	UDINT	Eingang: Obere Koordinate
nWidth	UDINT	Eingang: Breite
nHeight	UDINT	Eingang: Höhe
sFullName	STRING[nBRB_FILE_NAME_CHAR_MAX]	Eingang: Pfad zur Anbindung
nBackgroundColor	UINT	Zur Anbindung an den Datenpunkt
nStatus	UINT	Zur Anbindung an den Datenpunkt

Die Angaben nWidth und nHeight müssen die Breite bzw. Höhe des Drawbox-Controls enthalten, damit diese vor dem eigentlichen Zeichnen korrekt gelöscht werden kann.
Der Name der Drawbox wird unbedingt zur Referenzierung benötigt.
Die Syntax ist folgende: „Seitenname/Layername/Controlname“. Nur wenn diese Bezeichnung korrekt ist, kann in diese Drawbox gezeichnet werden (siehe AS-Hilfe „VisApi.VA_Attach“).
Mit der BackgroundColor wird die Drawbox vor dem Zeichnen gelöscht.

Alle anderen Angaben werden nicht benötigt.

BrbVc2dModelPadding_TYP		Einrückung des Modells
nLeft	DINT	Eingang: Einrückung Links
nTop	DINT	Eingang: Einrückung Oben

Das Padding legt die Einrückung des Modells in der Drawbox fest. So kann es z.B. mittig platziert werden.

rScaling gibt die Skalierung für das gesamte Modell an. So kann das Modell vergrößert (>1.0) oder verkleinert (<1.0) werden.
Achtung: Ein Skalier-Wert von 0.0 ist unzulässig und wird von der Draw-Funktion auf 1.0 korrigiert!

`pElements` muss auf ein Array von Elementen zeigen, welche das Modell definieren.
`nElementIndexMax` gibt den größten Index dieses Arrays an.

4.3.1.1.2 Callbacks

Hinweis: Diese Funktionalität ist aufgrund von Funktionszeigern nur in ANSI-C nutzbar, aber nicht in IEC-Sprachen (siehe Punkt Hinweise zu StructuredText und anderen IEC-Sprachen)

BrbVc2dModelCfgCallbacks_TYP		Konfiguration der Aufrufe
pCallbackAfterClear	UDINT	Eingang: Funktions-Zeiger für Aufruf nach Löschen der Drawbox
pCallbackBeforeElement	UDINT	Eingang: Funktions-Zeiger für Aufruf vor Zeichnen eines Elements
pCallbackAfterElement	UDINT	Eingang: Funktions-Zeiger für Aufruf nach Zeichnen eines Elements
pCallbackAfterModel	UDINT	Eingang: Funktions-Zeiger für Aufruf nach Zeichnen aller Elemente

Ein Callback ist ein Aufruf einer vom Anwender geschriebenen Funktion während des Zeichnens.
Er arbeitet mit sogenannten Funktions-Zeigern. Dabei wird die Adresse einer Funktion übergeben, welche der Anwender selbst schreibt. Lediglich die Signatur, also die Anzahl, Reihenfolge und die Datentypen der Argumente sind dabei vorgeschrieben. Der Inhalt der Funktion bleibt vollkommen dem Anwender überlassen.
Es gibt 4 verschiedene Callbacks (siehe oben), welche während des Zeichnens des Modells aufgerufen werden können.
Für jeden Callback gibt es ein Muster der Signatur in der Datei „BrbVc2dCallbackTemplates.c“ in der Bibliothek:

```
unsigned short BrbVc2dCallbackAfterClear(struct BrbVc2dModel_TYP* pModel)

unsigned short BrbVc2dCallbackBeforeElement(struct BrbVc2dModel_TYP* pModel, UDINT nElementIndex)

unsigned short BrbVc2dCallbackAfterElement(struct BrbVc2dModel_TYP* pModel, UDINT nElementIndex)

unsigned short BrbVc2dCallbackAfterModel(struct BrbVc2dModel_TYP* pModel)
```

Manche Callbacks werden während des Zeichnens mehrmals aufgerufen, so z.B. nach dem Zeichnen jeden Elements. Über Argumente werden aktuelle Werte übergeben, z.B. der aktuelle Element-Index.
Im Callback kann auf das gesamte Element-Array zugegriffen werden.
ACHTUNG: Es sollten aber keine Werte verändert werden!

Soll ein Callback aktiviert werden, so ist dessen Adresse in die obige Struktur einzutragen.
Beispiel:

Es wird eine Funktion angelegt, welche dem Muster entspricht:

```
unsigned short Robot2dCallbackAfterModel(struct BrbVc2dModel_TYP* pModel)
{
	// Anwender-Code
	return 0;
}
```

Vor dem Aufruf der Modell-Zeichen-Funktion wird die Adresse des Callbacks übergeben

```
Model.Cfg.Callbacks.pCallbackAfterModel = (UDINT)&Robot2dCallbackAfterModel;
```

Innerhalb des Callbacks kann dann mit Zeichenfunktionen die visuelle Ausgabe erweitert werden, z.B. Texte oder zusätzliche Linien eingezeichnet werden.
Der Rückgabewert der Funktion ist egal.
ACHTUNG: Das Koordinaten-System bezieht sich auf die Modell-Drawbox, weil diese noch referenziert ist.

4.3.1.1.3 pTag

Dieser Zeiger wird von der Funktion nicht benutzt. Er kann vom Anwender als Zeiger auf Benutzer-Daten gesetzt und dann in den Callbacks verwendet werden.

4.3.1.2 Interne Daten

BrbVc2dModelIntern_TYP		Interne Variablen des Modells
nAccessStatus	UINT	Intern: Status der Access-Funktion
nAttachStatus	UINT	Intern: Status der Attach-Funktion

Die Daten werden nur intern verwendet.

4.3.2 Hauptfunktion zum Zeichnen des Modells

4.3.2.1 BrbVc2dDrawModel

```
unsigned short BrbVc2DrawModel(struct BrbVc2dModel_TYP* pModel, struct BrbVc4General_TYP* pGeneral)
```

Argumente:

`struct BrbVc2dModel_TYP* pModel`
Zeiger auf das Modell

`struct BrbVc4General_TYP* pGeneral`
Zeiger auf die Instanz von „`BrbVc4General_TYP`“

Rückgabe:

`UINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Diese Funktion zeichnet das Modell nach Angaben der Modell-Struktur. Dabei werden die Elemente laut der Reihenfolge im Element-Array gezeichnet.
Sie sollte zyklisch aufgerufen werden.

Achtung: Es ist ein Zeiger auf die Struktur `General` zu übergeben. Damit diese korrekt befüllt ist, muss vorher die zugehörige Funktion `BrbVc4HandleGeneral` aus der Bibliothek `BrbLibVc4` aufgerufen werden. Siehe dazu auch die Hilfe von `BrbLibVc4`.

4.4 Transformationen (Dynamisierung)

Eine Transformation beschreibt eine Änderung eines Elements oder des ganzen Modells. Möglich sind:
-Skalierung (Vergrößern/Verkleinern)
-Translation (Verschiebung)
-Rotation (Drehung um einen beliebigen Drehpunkt).

Diese reichen aus, um auch komplexe Bewegungs-Abhängigkeiten darzustellen.
Es gibt eine Enumeration dazu:

BrbVc2dTransform_ENUM	Transformations-Arten
eBRBVC2D_TRANSFORM_SCALING	0=Skalierung
eBRBVC2D_TRANSFORM_TRANSLATE	1=Verschiebung
eBRBVC2D_TRANSFORM_ROTATE	2=Drehung

Durch Funktionen ist es möglich, ein Element (oder auch mehrere) entsprechend zu transformieren.

In dieser Bibliothek sind 2 Möglichkeiten implementiert, eine Transformation durchzuführen: Sequentielle oder Matrix-Transformation.
Beide haben Vor- und Nachteile (siehe unten), können aber auch gemischt verwendet werden, weil sie ja lediglich die Element-Koordinaten und -Parameter umrechnen, nur eben mit unterschiedlichen Konzepten.

4.4.1 Sequentielle Transformation

Bei dieser Art der Transformation wird jedes Element durch hintereinanderliegende Aufrufe der Transformations-Funktionen dynamisiert.
Diese Art ist sehr einfach in der Implementierung, benötigt aber vor allem bei vielen Elementen und Transformationen mehr Performance als die weiter unten beschriebene Matrix-Transformation. Sie eignet sich daher gut für sehr kleine Modelle.

4.4.1.1 Skalierung

4.4.1.1.1 BrbVc2dScaleElement

```
unsigned short BrbVc2dScaleElement(struct BrbVc2dElement_TYP* pElement, float rX, float rY)
```

Argumente:

`struct BrbVc2dElement_TYP* pElement`
Zeiger auf das Element

`REAL rX`
Skalierung in X-Richtung

`REAL rY`
Skalierung in Y-Richtung

Rückgabe:

`UINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Das übergebene Element wird mit den entsprechenden Werten in X- bzw. Y-Richtung skaliert. Ein Wert über 1.0 vergrößert, ein Wert unter 1.0 verkleinert das Element.
Achtung: Ein Skalier-Wert von 0.0 ist unzulässig und wird von der Funktion auf 1.0 korrigiert!
Hinweis: Da alle Punkte des Elements skaliert werden, verändert sich dabei nicht nur dessen Größe, sondern auch dessen Position!

4.4.1.2 Verschiebung

4.4.1.2.1 BrbVc2dTranslateElement

```
unsigned short BrbVc2dTranslateElement(struct BrbVc2dElement_TYP* pElement, float rX, float rY)
```

Argumente:

`struct BrbVc2dElement_TYP* pElement`
Zeiger auf das Element

`REAL rX`
Verschiebung in X-Richtung

`REAL rY`
Verschiebung in Y-Richtung

Rückgabe:

`UINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Das übergebene Element wird um die entsprechenden Werte in X- bzw. Y-Richtung verschoben.

4.4.1.3 Drehung

4.4.1.3.1 BrbVc2dRotateElement

```
unsigned short BrbVc2dRotateElement(struct BrbVc2dElement_TYP* pElement, float rX, float rY,
float rAngle)
```

Argumente:

`struct BrbVc2dElement_TYP* pElement`
Zeiger auf das Element

`REAL rX`
X-Koordinate des Drehpunkts

`REAL rY`
Y-Koordinate des Drehpunkts

`REAL rAngle`
Drehwinkel in [°] im Uhrzeigersinn

Rückgabe:

`UINT`
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`

Beschreibung:
Das übergebene Element wird um den Drehpunkt mit dem Winkel im Uhrzeigersinn gedreht.

4.4.2 Matrix-Transformation

Dies ist eine alternative Art der Transformation. Dabei werden die benötigten Berechnungen mithilfe einer sogenannten Matrix (= Parameter-Block) ausgeführt. Der Vorteil ist, dass 1 Matrix beliebig viele hintereinander folgende Transformationen aufnehmen kann. Die Umrechnung eines mehrfach zu transformierenden Elements muss dann einmalig mit dieser Matrix geschehen.
Diese Art ist ein wenig komplexer in der Implementierung, benötigt aber vor allem bei vielen Elementen und Transformationen sehr viel weniger Performance als die weiter oben beschriebene sequentielle Transformation. Sie eignet sich daher gut für große Modelle.
Hinweis: Die Mehrzahl von Matrix wird als Matrizen bezeichnet.

Begonnen wird mit der Identitäts-Matrix. In ihr sind die Parameter so gesetzt, dass keine Transformation geschieht. Soll eine Transformation (Skalierung, Verschiebung oder Drehung) hinzugefügt werden, so wird eine entsprechende Transformations-Matrix durch Funktionsaufruf erstellt und diese mit der ursprünglichen Matrix durch Multiplikation verkettet. So können beliebig viele Transformationen in einer Matrix verkettet werden.

4.4.2.1 Matrix-Struktur

BrbVc2dMatrix_TYP		
rM11	REAL	Matrix-Parameter
rM12	REAL	Matrix-Parameter
rM13	REAL	Matrix-Parameter
rM21	REAL	Matrix-Parameter
rM22	REAL	Matrix-Parameter
rM23	REAL	Matrix-Parameter
rM31	REAL	Matrix-Parameter
rM32	REAL	Matrix-Parameter
rM33	REAL	Matrix-Parameter
rM41	REAL	Matrix-Parameter
rM42	REAL	Matrix-Parameter
rM43	REAL	Matrix-Parameter

Die Bedeutung der einzelnen Matrix-Parameter ist für den Anwender nicht wichtig.
Für Mathematiker: Für die Transformation eines Punktes im 2-dimensionalen Raum werden eigentlich nur die Parameter M11..M33 benötigt. Da es aber Elemente gibt, bei denen nicht alle Punkte, sondern Konstruktions-Parameter angegeben sind, werden die zusätzlichen Parameter M41..M43 benötigt. So werden

bei einer Ellipse nicht alle Punkte gespeichert (das wäre zu speicher-intensiv), sondern nur der Mittelpunkt. Die restlichen Daten werden als Konstruktions-Parameter (z.B. X- und Y-Skalierung bzw. der Drehwinkel) hinterlegt, welche nicht durch herkömmliche Matrizen-Konzepte umgerechnet werden können. Zur Umrechnung dieser Konstruktions-Parameter werden dann M41..M43 verwendet.

4.4.2.2 Transformations-Funktionen

4.4.2.2.1 BrbVc2dGetIdentityMatrix

```
unsigned short BrbVc2dGetIdentityMatrix(struct BrbVc2dMatrix_TYP* pMatrix)
```

Argumente:

```
struct BrbVc2dMatrix_TYP* pMatrix
```
Zeiger auf die Matrix

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Die übergebene Matrix wird mit den Werten einer Identitäts-Matrix besetzt.
Diese enthält keine Transformation und wird üblicherweise als Start-Matrix für Verkettungen benötigt.

4.4.2.2.2 BrbVc2dGetScalingMatrix

```
unsigned short BrbVc2dGetScalingMatrix(struct BrbVc2dMatrix_TYP* pMatrix, float rX, float rY)
```

Argumente:

```
struct BrbVc2dMatrix_TYP* pMatrix
```
Zeiger auf die Matrix
```
REAL rX
```
Skalierung in X-Richtung
```
REAL rY
```
Skalierung in Y-Richtung

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Die übergebene Matrix wird mit den entsprechenden Werten zur Skalierung besetzt. Durch Multiplikation mit einer bestehenden Matrix (siehe unten) kann die Transformation verkettet werden.
Achtung: Ein Skalier-Wert von 0.0 ist unzulässig und wird von der Funktion auf 1.0 korrigiert!

4.4.2.2.3 BrbVc2dGetTranslationMatrix

```
unsigned short BrbVc2dGetTranslationMatrix(struct BrbVc2dMatrix_TYP* pMatrix, float rX, float rY)
```

Argumente:

```
struct BrbVc2dMatrix_TYP* pMatrix
```
Zeiger auf die Matrix
```
REAL rX
```
Verschiebung in X-Richtung
```
REAL rY
```
Verschiebung in Y-Richtung

Rückgabe:

```
UINT
        eBRB_ERR_OK = 0
        eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Die übergebene Matrix wird mit den entsprechenden Werten zur Translation besetzt. Durch Multiplikation mit einer bestehenden Matrix (siehe unten) kann die Transformation verkettet werden.

4.4.2.2.4 BrbVc2dGetRotationMatrix

```
unsigned short BrbVc2dGetRotationMatrix(struct BrbVc2dMatrix_TYP* pMatrix, float rX, float rY,
float rAngle)
```

Argumente:

`struct BrbVc2dMatrix_TYP* pMatrix`
Zeiger auf die Matrix

`REAL rX`
X-Koordinate des Drehpunkts

`REAL rY`
Y-Koordinate des Drehpunkts

`REAL rAngle`
Drehwinkel in [°] im Uhrzeigersinn

Rückgabe:

`UINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Die übergebene Matrix wird mit den entsprechenden Werten zur Rotation besetzt. Durch Multiplikation mit einer bestehenden Matrix (siehe unten) kann die Transformation verkettet werden.

4.4.2.2.5 BrbVc2dMultiplyMatrices

```
unsigned short BrbVc2dMultiplyMatrices(struct BrbVc2dMatrix_TYP* pMatrix1, struct
BrbVc2dMatrix_TYP* pMatrix2)
```

Argumente:

`struct BrbVc2dMatrix_TYP* pMatrix1`
Zeiger auf die 1. Matrix

`struct BrbVc2dMatrix_TYP* pMatrix2`
Zeiger auf die zu verkettende 2. Matrix

Rückgabe:

`UINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Die 1. Matrix wird mit der 2. Matrix multipliziert und somit deren Transformationen verkettet. Das Ergebnis wird wieder in der 1. Matrix abgelegt. So können beliebig viele Transformationen in nur 1 Matrix abgebildet werden.

4.4.2.2.6 BrbVc2dTransformElement

```
unsigned short BrbVc2dTransformElement(struct BrbVc2dElement_TYP* pElement, struct
BrbVc2dMatrix_TYP* pMatrix)
```

Argumente:

`struct BrbVc2dPoint_TYP* pPoint`
Zeiger auf den Drehpunkt

`struct BrbVc2dMatrix_TYP* pMatrix`
Zeiger auf die Matrix

Rückgabe:

`UINT`

```
eBRB_ERR_OK = 0
eBRB_ERR_NULL_POINTER = 50000
```

Beschreibung:

Die Punkte und Konstruktions-Parameter des Elements werden mithilfe einer Matrix transformiert.

4.4.2.3 Hilfsfunktionen

4.4.2.3.1 BrbVc2dSetMatrix

```
unsigned short BrbVc2dSetMatrix(struct BrbVc2dMatrix_TYP* pMatrix, float rM11, float rM12, float
rM13, float rM21, float rM22, float rM23, float rM31, float rM32, float rM33, float rM41, float
rM42, float rM43)
```

Argumente:

`struct BrbVc2dMatrix_TYP* pMatrix`
Zeiger auf die Matrix

`REAL rM11..rM43`
Matrix-Parameter

Rückgabe:

`UINT`
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`

Beschreibung:

Die übergebene Matrix wird mit den entsprechenden Werten besetzt.
Die Funktion wird normalerweise nicht applikativ benötigt, sondern nur intern verwendet.

4.4.2.3.2 BrbVc2dTransformPoint

```
unsigned short BrbVc2dTransformPoint(struct BrbVc2dPoint_TYP* pPoint, struct BrbVc2dMatrix_TYP*
pMatrix)
```

Argumente:

`struct BrbVc2dPoint_TYP* pPoint`
Zeiger auf den Drehpunkt

`struct BrbVc2dMatrix_TYP* pMatrix`
Zeiger auf die Matrix

Rückgabe:

`UINT`
`eBRB_ERR_OK = 0`
`eBRB_ERR_NULL_POINTER = 50000`

Beschreibung:

Die Koordinaten des Punkts werden mithilfe einer Matrix transformiert.
Die Funktion wird normalerweise nicht applikativ benötigt, sondern nur intern verwendet.

5 Allgemeine Hinweise zur Dynamisierung eines Modells

5.1 Original und Transformation

Ein 2d-Modell kann den aktuellen Zustand eines Maschinen-Teils anzeigen. Dazu werden in jedem Zyklus die Elemente eines Modells aufgrund der Maschinen-Werte (z.B. Achs-Positionen) neu transformiert und dann gezeichnet.
Da sowohl die sequentielle als auch die Matrix-Transformierungs-Funktionen direkt die im Element enthaltenen Werte verändern, muss in jedem Zyklus vor der Transformation dafür gesorgt werden, dass die Elemente ihre Original-Werte enthalten.
Dazu gibt es zwei Varianten:

- Es gibt 2 Element-Arrays. Das Original wird einmalig mit den Elementen besetzt. In jedem Zyklus wird das Original auf eine Kopie übertragen, dass dann transformiert und zum Zeichen verwendet wird.
- Es gibt nur ein Element-Array. Vor der Transformation werden die Elemente neu mit den Original-Werten besetzt.

5.2 Modell initialisieren

Das Original kann mithilfe der `BrbVc2dSetElementXXX`-Funktionen (siehe oben) zur Laufzeit erstellt werden.

5.3 Transformation der Modell-Elemente

Durch die Transformations-Funktionen kann jedes Element verändert werden. Besteht ein Bauteil aus mehreren Elementen, muss die Transformation auf alle Elemente dieses Bauteils angewendet werden (Beispiel siehe unten).

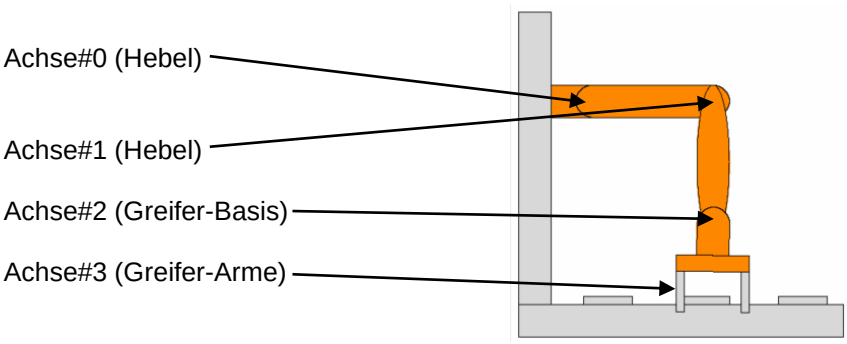
5.4 Zykluszeiten

Bei Vc4 wird das Display durch einen VNC-Viewer angezeigt, der sich das anzuzeigende Bild vom VNC-Server der SPS holt. Dies geschieht in einem bestimmten Intervall, das beim VNC-Server (‚Refresh rate') angegeben werden kann.
Es macht natürlich keinen Sinn, dass Modell in einem schnelleren Intervall zu zeichnen als das Bild aufgefrischt wird. Dies führt nur zur unnötigen Belastung der CPU.
Es reicht daher aus, die Transformierung und die Zeichen-Funktion des Modells nur in jedem x-ten Zyklus auszuführen, abgestimmt auf das Intervall der VNC-Verbindung. Bequemerweise kann dazu der schon vorhandene Zähler `General.nRedrawCounter` aus der Bibliothek BrbLibVc4 verwendet werden.

Hinweis: Da gewöhnlicherweise die RefreshRate eines VNC-Serves nicht sehr klein ist (>= 100ms), werden schnelle Bewegungen (also Achs-Positions-Änderungen) auch nicht flüssig, sondern meist ‚ruckelig' dargestellt. Die animierte 2d-Darstellung ist ab einem gewissen Grad daher nicht mehr sinnvoll.

6 Beispiel 2d-Roboter

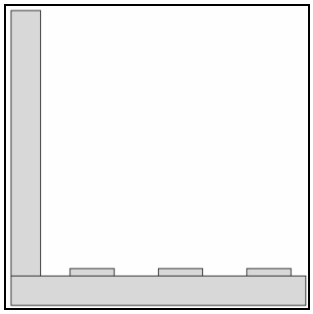
Im Demo-Projekt ist beispielhaft ein 2d-Roboter implementiert, der 4 Achsen besitzt:



Die meisten Bauteile bestehen aus mehreren Elementen.

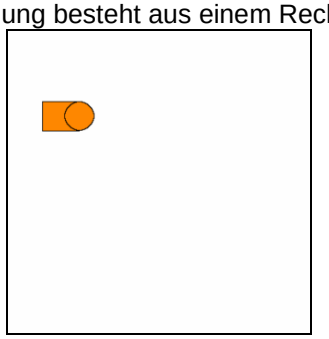
6.1 Aufbau

Die Plattform besteht aus mehreren, nicht dynamisierten Rechtecken:



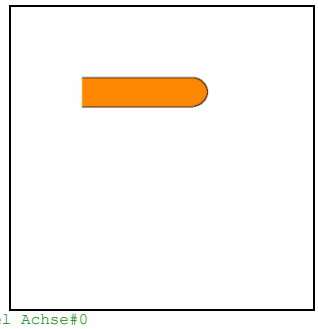
```
// Vertikale Plattform
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 10, 10, 40, 400, 0, 0, 0, 0, 250, 1);
// Horizontale Plattform
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 10, 370, 400, 40, 0, 0, 0, 0, 250, 1);
// Box-Position #0
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 90, 360, 60, 10, 0, 0, 0, 0, 250, 1);
// Box-Position #1
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 210, 360, 60, 10, 0, 0, 0, 0, 250, 1);
// Box-Position #2
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 330, 360, 60, 10, 0, 0, 0, 0, 250, 1);
```

Die Aufhängung besteht aus einem Rechteck und einem Kreis



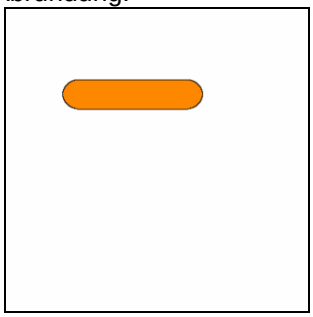
```
// Roboter Basis
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 50, 100, 50, 40, 0, 255, 0, 0, 253, 1);
BrbVc2dSetElementCircle(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 100, 120, 20, 0, 253, 1);
```

Der erste Hebel (Achse#0) besteht aus einem Kreis und einem Rechteck:

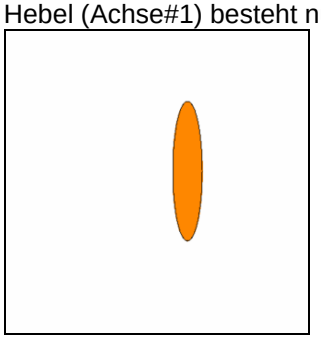


```
// Roboter Hebel Achse#0
Vis.PageVc2dRobot.nStartIndexAxis0 = nElementIndex;
BrbVc2dSetElementCircle(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 250, 120, 20, 0, 253, 1);
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 100, 100, 150, 40, 0, 255, 0, 255, 253, 1);
```

Das Rechteck wird nach dem Kreis gezeichnet, wodurch der linke Halbkreis überdeckt wird. Sind außerdem die linke und die rechte Randlinie des Rechtecks transparent, so entsteht auf beiden Seiten der Eindruck einer Abrundung:

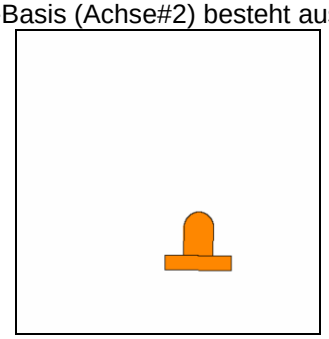


Der nächste Hebel (Achse#1) besteht nur aus einer Ellipse:



```
// Roboter Hebel Achse#1
Vis.PageVc2dRobot.nStartIndexAxis1 = nElementIndex;
BrbVc2dSetElementEllipse(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 250, 195, 20, 95, 0, 0, 253, 1);
```

Die Greifer-Basis (Achse#2) besteht aus einem Kreis und zwei Rechtecken:



```
// Roboter Greifarm-Basis Achse#2
Vis.PageVc2dRobot.nStartIndexAxis2 = nElementIndex;
BrbVc2dSetElementCircle(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 250, 270, 20, 0, 253, 1);
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 230, 270, 40, 40, 255, 0, 0, 0, 253, 1);
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 205, 310, 90, 20, 0, 0, 0, 0, 253, 1);
```

Auch hier wird durch die richtige Reihenfolge und der transparenten Rahmen-Linie der Eindruck einer Abrundung erzeugt.

Der Greifer (Achse#3) selbst besteht schließlich nur aus zwei Rechtecken:

```
// Roboter Greifarm-Greifer
Vis.PageVc2dRobot.nIndexGripper0 = nElementIndex;
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 205, 330, 10, 50, 0, 0, 0, 0, 250, 1);
Vis.PageVc2dRobot.nIndexGripper1 = nElementIndex;
BrbVc2dSetElementTetragon2(&Vis.PageVc2dRobot.ElementsOriginal[nElementIndex++], 285, 330, 10, 50, 0, 0, 0, 0, 250, 1);
Vis.PageVc2dRobot.nEndIndexRobot = nElementIndex-1;
```

Beim Erstellen des Originals werden sich bestimmte Indizes des Element-Arrays gemerkt (siehe Code oben), z.B. Start- und End-Index, die zu einer Achse gehören und deshalb später miteinander gleich transformiert werden müssen.

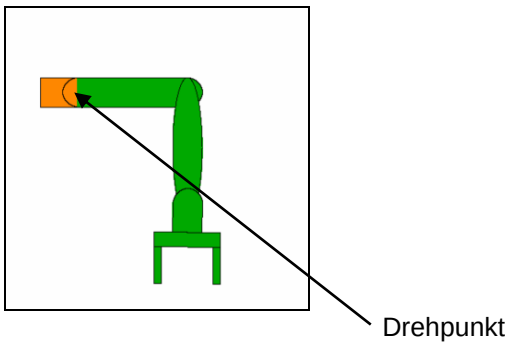
6.2 Dynamisierung durch Transformation

Die Transformation der Elemente geschieht in Schleifen, bei denen zusammengehörige Gruppen von Elementen gleich transformiert werden. Dabei helfen die zuvor gemerkten Indizes.
Zur Veranschaulichung sind beide Transformations-Konzepte (sequentiell und Matrix) implementiert. Durch eine Checkbox kann umgeschaltet werden.

6.2.1 Sequentielle Transformation

Zuerst werden alle Elemente ab Achse#0 um den Mittelpunkt des entsprechenden Kreises gedreht (betroffene Elemente sind grün markiert):

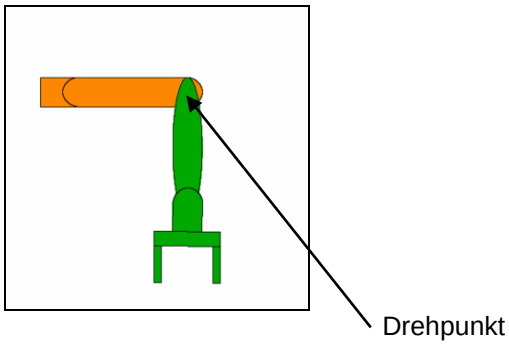
```
// Drehung Achse#0
for(nElementIndex=Vis.PageVc2dRobot.nStartIndexAxis0; nElementIndex<=Vis.PageVc2dRobot.nEndIndexRobot; nElementIndex++)
{
      BrbVc2dRotateElement(&Vis.PageVc2dRobot.ElementsTransformed[nElementIndex], 100, 120, 90.0 -
Vis.PageVc2dRobot.rAngleAxis0);
}
```



Dann werden alle Elemente ab Achse#1 gedreht. Als Drehpunkt wird der Mittelpunkt des zweiten Kreises benutzt, der ja schon um Achse#0 gedreht wurde:

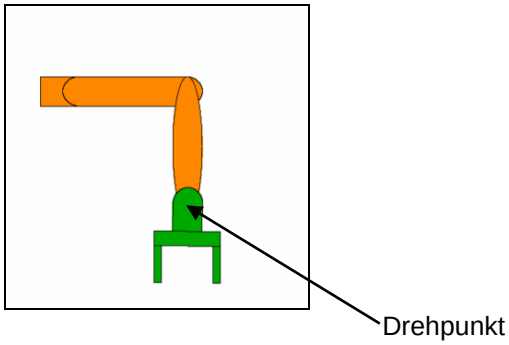
```
// Drehung Achse#1
pElement = &Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nStartIndexAxis0]; // Kreis-Element enthält Drehpunkt der
Achse
for(nElementIndex=Vis.PageVc2dRobot.nStartIndexAxis1; nElementIndex<=Vis.PageVc2dRobot.nEndIndexRobot; nElementIndex++)
{
```

```
      BrbVc2dRotateElement(&Vis.PageVc2dRobot.ElementsTransformed[nElementIndex], pElement->Points[0].rX, pElement->Points[0].rY,
180.0 - Vis.PageVc2dRobot.rAngleAxis1);
}
```



Als nächstes werden alle Elemente ab Achse#2 um den Mittelpunkt des entsprechenden Kreises gedreht:

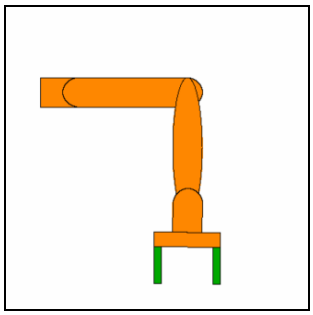
```
// Drehung Achse#2 (Winkel wird berechnet, so dass Greifer immer waagrecht steht)
pElement = &Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nStartIndexAxis1+1]; // Kreis-Element enthält Drehpunkt der
Achse
for(nElementIndex=Vis.PageVc2dRobot.nStartIndexAxis2; nElementIndex<=Vis.PageVc2dRobot.nEndIndexRobot; nElementIndex++)
{
      BrbVc2dRotateElement(&Vis.PageVc2dRobot.ElementsTransformed[nElementIndex], pElement->Points[0].rX, pElement->Points[0].rY,
+Vis.PageVc2dRobot.rAngleAxis0 + Vis.PageVc2dRobot.rAngleAxis1 + 90.0);
}
```



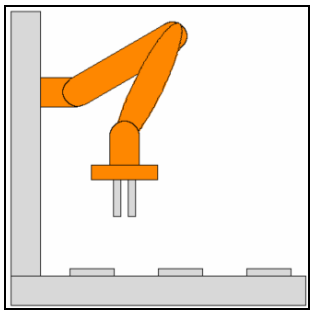
Als Besonderheit wird hier keine Achsposition zur Transformation verwendet. Vielmehr wird aus den Positionen der beiden vorigen Achsen der Winkel so berechnet, dass der Greifer immer waagrecht steht.

Zuletzt werden die Greifer-Elemente so verschoben, dass die beiden Greifer-Arme aufeinander zufahren.

```
// Greifer
BrbVc2dTranslateElement(&Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nIndexGripper0], +Vis.PageVc2dRobot.rPosGripper,
0);
BrbVc2dTranslateElement(&Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nIndexGripper1], -Vis.PageVc2dRobot.rPosGripper,
0);
```



Auf diese Weise kann aufgrund der Achs-Positionen der Roboter in allen Stellungen gezeichnet werden:



6.2.2 Matrix-Transformation

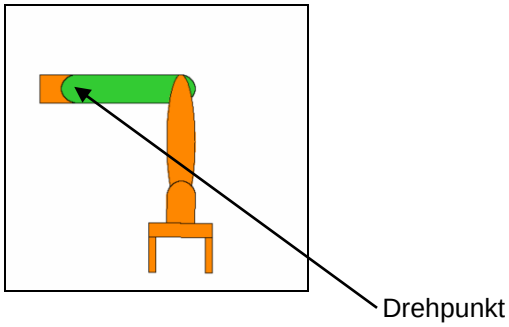
Diese ist in ähnlicher Weise implementiert wie die sequentielle Transformation. Auch hier werden die Elemente gruppenweise transformiert.
Allerdings wird dazu eine Hauptmatrix generiert und diese schrittweise mit immer mehr Transformationen verkettet. Zwischendurch werden dann entsprechende Elementgruppen damit transformiert.

Zunächst das Erstellen der Hauptmatrix und der temporären Matrix für Verkettungen:

```
BrbVc2dGetIdentityMatrix(&Matrix2d); // Hauptmatrix zum Verketten
BrbVc2dMatrix_TYP TransformMatrix;
```

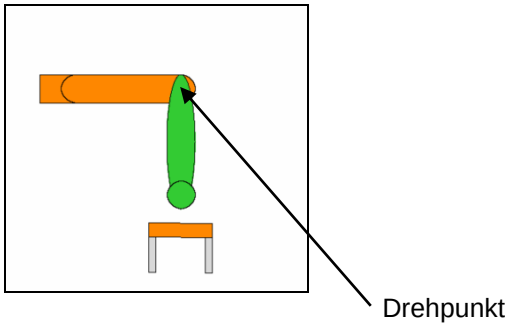
Dann werden alle Elemente der Achse#0 um den Mittelpunkt des entsprechenden Kreises gedreht (betroffene Elemente sind grün markiert):

```
// Drehung Achse#0
BrbVc2dGetRotationMatrix(&TransformMatrix, 100, 120, 90.0 - Vis.PageVc2dRobot.rAngleAxis0);
BrbVc2dMultiplyMatrices(&Matrix2d, &TransformMatrix);
for(nElementIndex=Vis.PageVc2dRobot.nStartIndexAxis0; nElementIndex<Vis.PageVc2dRobot.nStartIndexAxis1; nElementIndex++)
{
      BrbVc2dTransformElement(&Vis.PageVc2dRobot.ElementsTransformed[nElementIndex], &Matrix2d);
}
```



Dann werden die Elemente der Achse#1 gedreht. Als Drehpunkt wird der Mittelpunkt des zweiten Kreises benutzt, der ja schon um Achse#0 gedreht wurde. Da die bestehende Hauptmatrix mit der neuen Drehung verkettet wird, ist die erste Drehung noch mit drin.

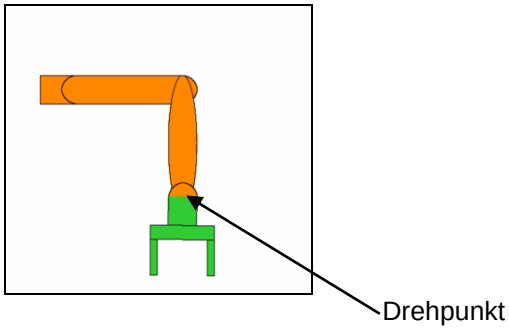
```
pElement = &Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nStartIndexAxis0]; // Kreis-Element enthält Drehpunkt der
Achse
BrbVc2dGetRotationMatrix(&TransformMatrix, pElement->Points[0].rX, pElement->Points[0].rY, 180.0 -
Vis.PageVc2dRobot.rAngleAxis1);
BrbVc2dMultiplyMatrices(&Matrix2d, &TransformMatrix);
// 1. Element der Achse#2 (Kreis) wird mitgedreht, da er als Drehpunkt für Achse#2 verwendet wird
for(nElementIndex=Vis.PageVc2dRobot.nStartIndexAxis1; nElementIndex<=Vis.PageVc2dRobot.nStartIndexAxis2; nElementIndex++)
{
      BrbVc2dTransformElement(&Vis.PageVc2dRobot.ElementsTransformed[nElementIndex], &Matrix2d);
}
```



Als Besonderheit muss hier der Kreis der Greifer-Basis auch schon transformiert werden, da sein Mittelpunkt für die nächste Matrix gebraucht wird.

Als nächstes werden alle weiteren Elemente ab Achse#2 um den Mittelpunkt des entsprechenden Kreises gedreht:

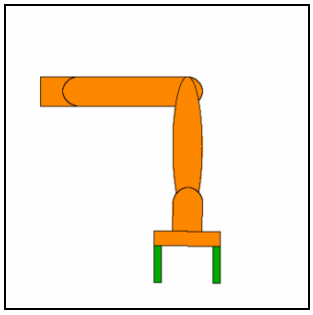
```
// Drehung Achse#2 (Winkel wird berechnet, so dass Greifer immer waagrecht steht)
pElement = &Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nStartIndexAxis1+1]; // Kreis-Element enthält Drehpunkt der
Achse
BrbVc2dGetRotationMatrix(&TransformMatrix, pElement->Points[0].rX, pElement->Points[0].rY, +Vis.PageVc2dRobot.rAngleAxis0 +
Vis.PageVc2dRobot.rAngleAxis1 + 90.0);
BrbVc2dMultiplyMatrices(&Matrix2d, &TransformMatrix);
// Kreis der Achse#2 muss nicht mehr enthalten sein, weil er bei Achse#1 schon mitgedreht wurde
// Die Greifer werden mitgedreht
for(nElementIndex=Vis.PageVc2dRobot.nStartIndexAxis2+1; nElementIndex<=Vis.PageVc2dRobot.nEndIndexRobot; nElementIndex++)
{
      BrbVc2dTransformElement(&Vis.PageVc2dRobot.ElementsTransformed[nElementIndex], &Matrix2d);
}
```



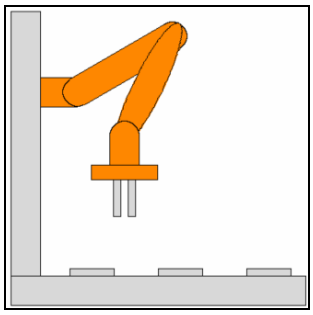
Als Besonderheit wird hier keine Achsposition zur Transformation verwendet. Vielmehr wird aus den Positionen der beiden vorigen Achsen der Winkel so berechnet, dass der Greifer immer waagrecht steht.

Zuletzt werden die Greifer-Elemente so verschoben, dass die beiden Greifer-Arme aufeinander zufahren.

```
// Greifer
BrbVc2dGetTranslationMatrix(&TransformMatrix, +Vis.PageVc2dRobot.rPosGripper, 0);
BrbVc2dTransformElement(&Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nIndexGripper0], &TransformMatrix);// Muss nur
verschoben werden, da bereits vorher mitgedreht
BrbVc2dGetTranslationMatrix(&TransformMatrix, -Vis.PageVc2dRobot.rPosGripper, 0);
BrbVc2dTransformElement(&Vis.PageVc2dRobot.ElementsTransformed[Vis.PageVc2dRobot.nIndexGripper1], &TransformMatrix);// Muss nur
verschoben werden, da bereits vorher mitgedreht
```



Das Ergebnis ist dasselbe wie bei der sequentiellen Transformation. Aufgrund der Achs-Positionen wird der Roboter in allen Stellungen gezeichnet:



Durch die Verkettung der Matrizen müssen weniger Berechnungen ausgeführt werden (die einzelnen Schleifen gehen jeweils nicht über so viele Elemente und die darin aufgerufene Funktion enthält keine Winkel-Funktionen Sinus und Cosinus). Deshalb ist die CPU-Auslastung geringer.

6.3 Automatik-Betrieb

Zu Demonstrations-Zwecken wurden noch zwei Boxen (blau und grün) in das Modell integriert, welche nur beim Automatik-Betrieb gezeichnet werden. Der Automatik-Betrieb wurde in einen eigenen Task ausgelagert, welcher tatsächlich nur die Achs-Positionen verändert. Diese werden dann benutzt, um das Modell zu transformieren und den aktuellen Zustand anzuzeigen. Als Folge sieht man eine animierte Darstellung des Roboters, der die zwei Boxen abwechselnd auf den Box-Positionen austauscht: