DigiWEB
C-Schnittstelle

Digitronic Automationsanlagen GmbH

Steinbeisstraße 3 • D - 72636 Frickenhausen • Tel. +49 7022 40590-0 • Fax -10
Auf der Langwies 1 • D - 65510 Hünstetten-Wallbach • Tel. +49 6126 9453-0 • Fax -42
Internet: http://www.digitronic.com • E-Mail: mail@digitronic.com

Inhalt

1 Einleitung

Dieses Handbuch beschreibt die Vorgehensweise, mit der Applikationen auf Basis des DigiWEB-Betriebssystems programmiert werden können.
Die Programmierung der Software erfolgt in der Regel auf einem WindowsPC mit Visual-Studio. Hier wird die Hardware eines DigiWEB auf dem PC emuliert. Das hat den Vorteil, dass die gesamte Entwicklung der Software am PC stattfindet, dessen Debug- und Entwicklungstools sehr einfach zu bedienen sind.
Nach dem Fertigstellen der Applikation müssen die Quellen nur noch für die entsprechende Hardwarebasis kompiliert werden.

2 Voraussetzungen zur DigiWEB-Programmierung

2.1 Visual Studio 6

Zur Programmierung am PC muss mit Visual Studio in der Version 6 sp6 gearbeitet werden.

2.2 DigiWEB Library

Diese Library stellt die Betriebssystemfunktionen des DigiWEB zur Verfügung. Sie ist ausschließlich bei der Firma Digitronic Automationsanlagen erhältlich und wird in verschiedenen Versionen (je nach Hardwarebasis) ausgeliefert.

2.3 Programmierung mit C

Zur Programmierung auf Basis des DigiWEB-Betriebssystems sollten fundierte Kenntnisse in der Entwicklung mit der Programmiersprache C und Mikrocontrollern vorhanden sein.

2.4 Gnu Compiler

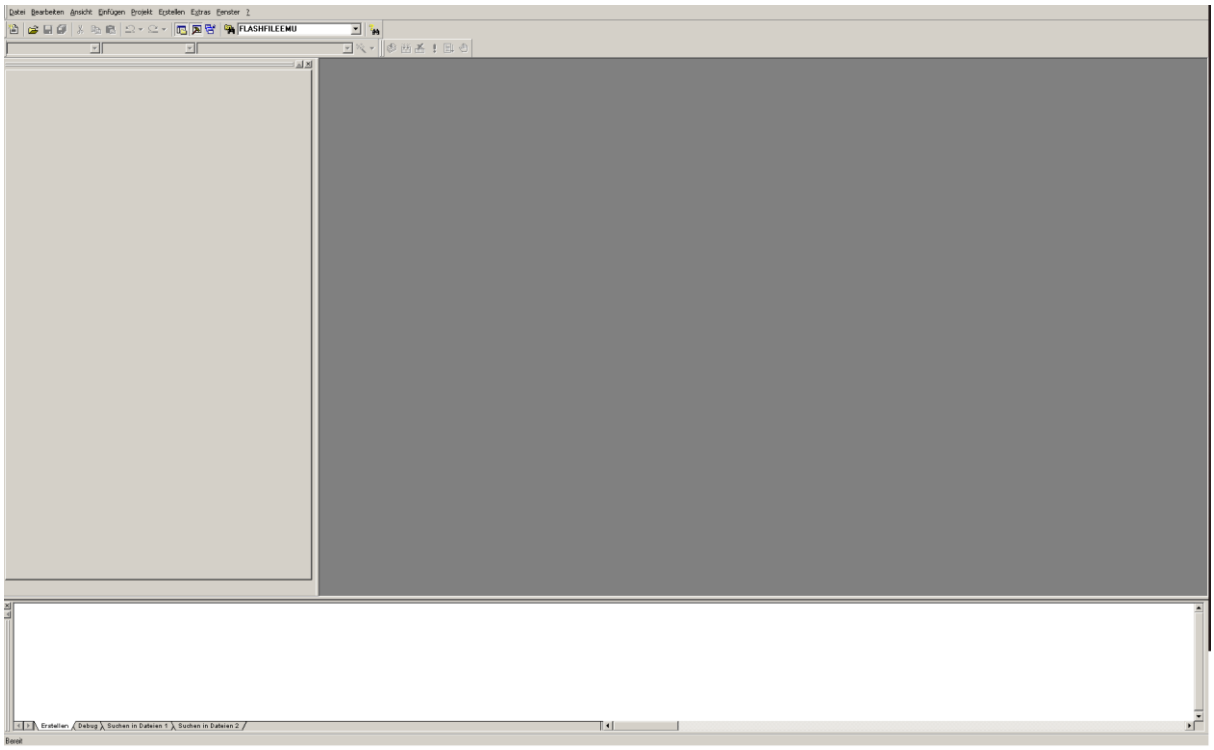
- Um die Applikation für die jeweilige Hardwarebasis zu kompilieren wird der gcc-Compiler von GNU in der Version .4.2.2 benötigt.
- Momentan werden folgende Prozessortypen unterstützt:
- SAMG920
- ARM91

2.5 Hardware Debugger

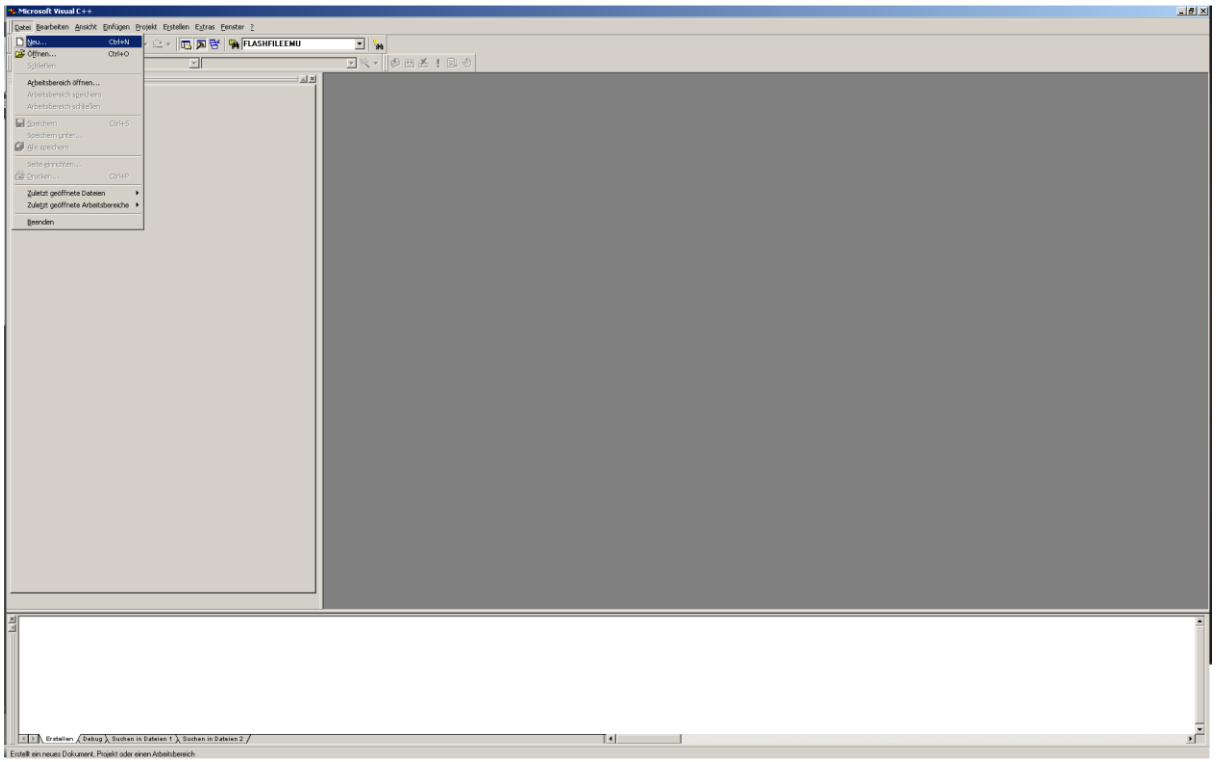
Um die Applikation direkt auf der Hardware zu Debuggen wird ein Debugger der Firma Lauterbach GmbH empfohlen.

3 Einrichten der Entwicklungsumgebung

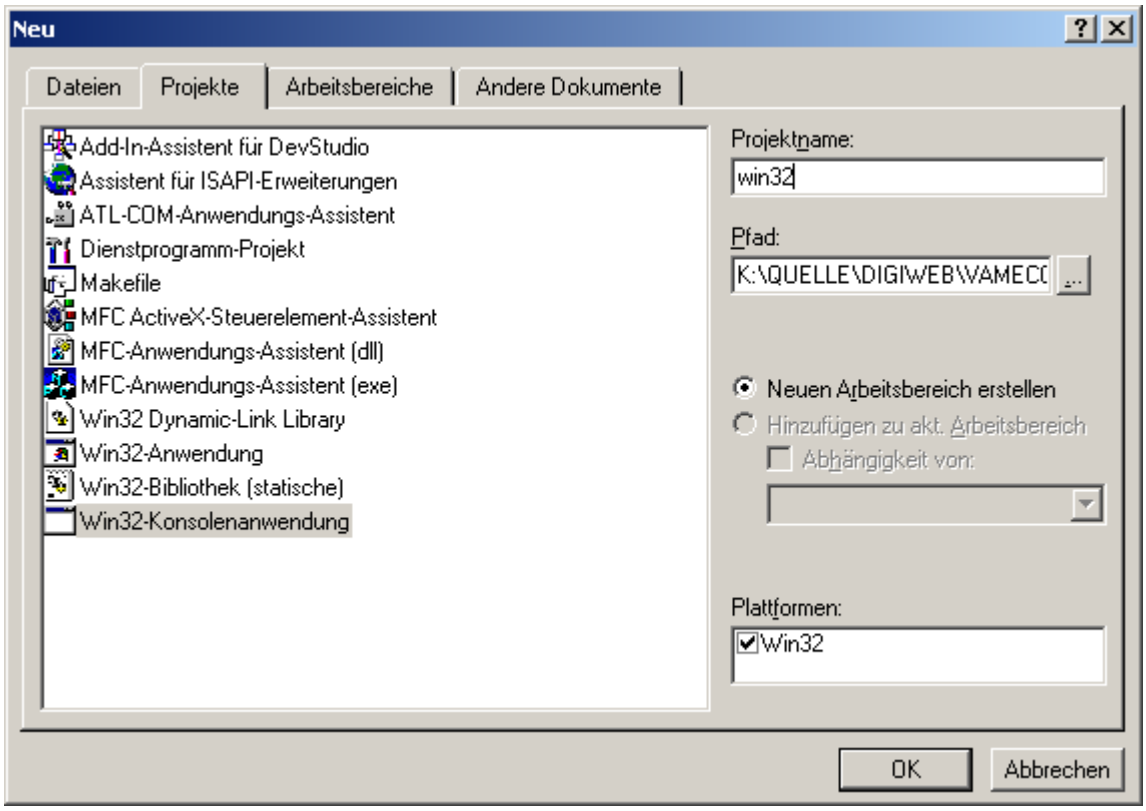
Starten Sie Visual Studio



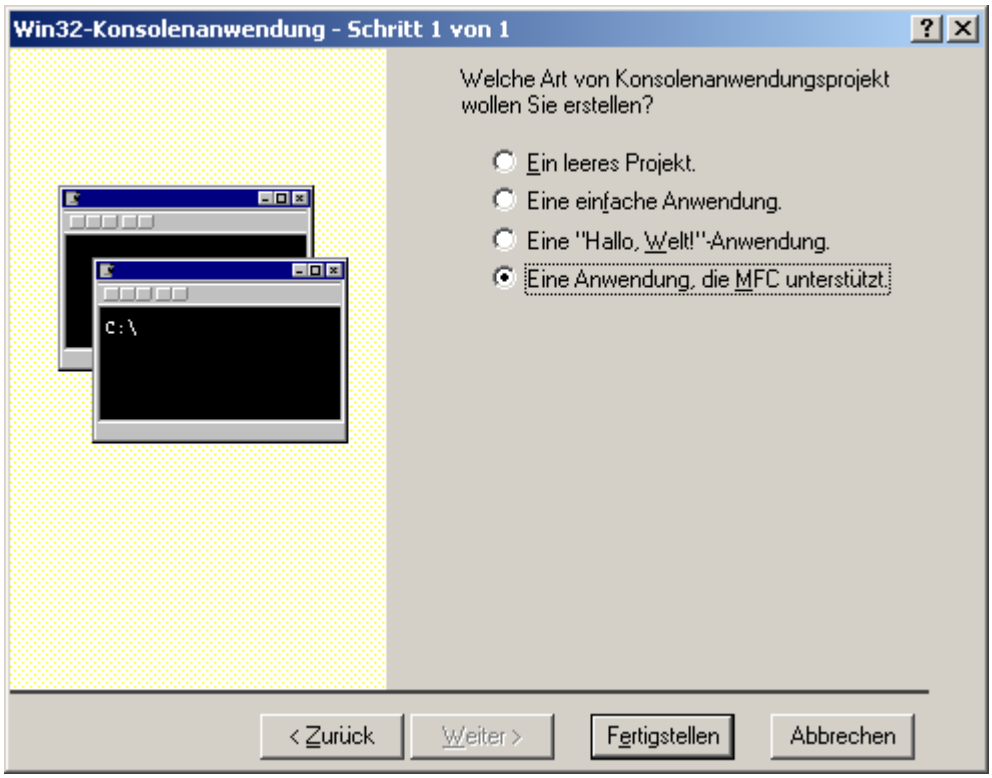
Gehen Sie zu „Datei → Neu“



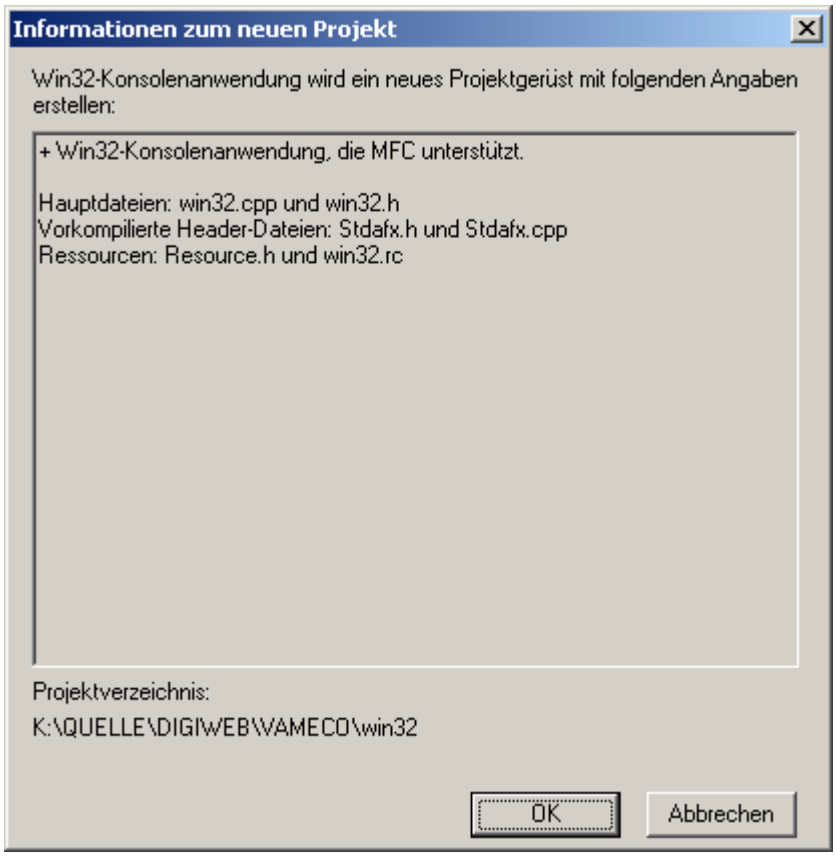
In dem sich öffnendem Fenster wählen Sie im Reiter Projekte „Win32-Konsolenanwendung“ und geben Sie im Eingabefeld „Projektname“ Ihren Projektnamen ein. Anschließend klicken Sie auf OK.



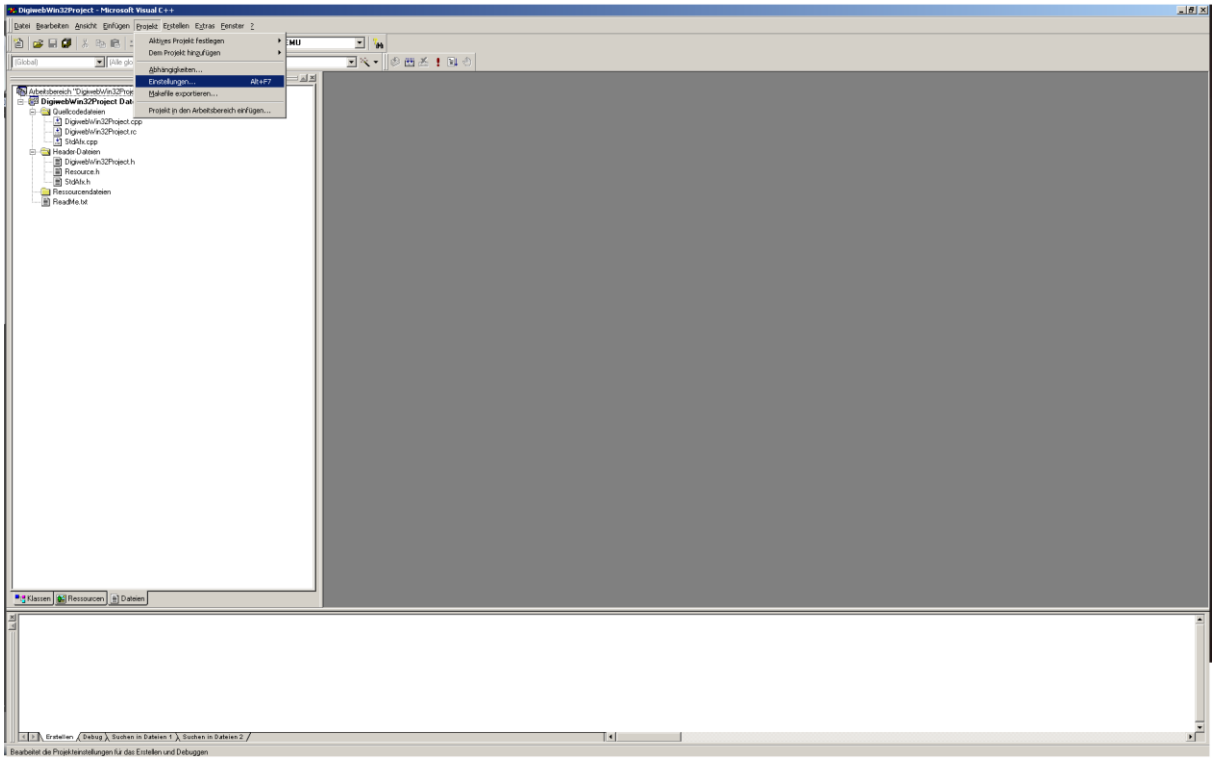
Im sich öffnenden Dialog wählen Sie „Eine Anwendung, die MFC unterstützt“.



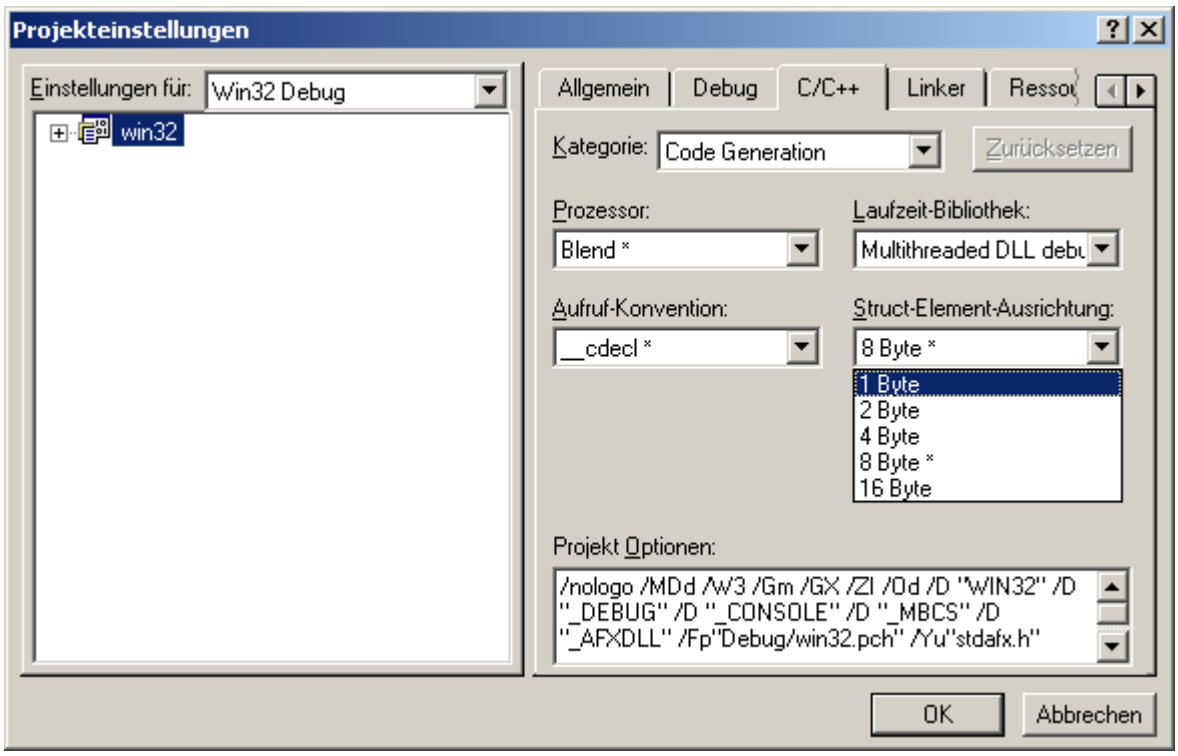
Daraufhin werden Informationen zu Ihrem gerade erstellten Projekt angezeigt. Diese müssen mit OK bestätigt werden.



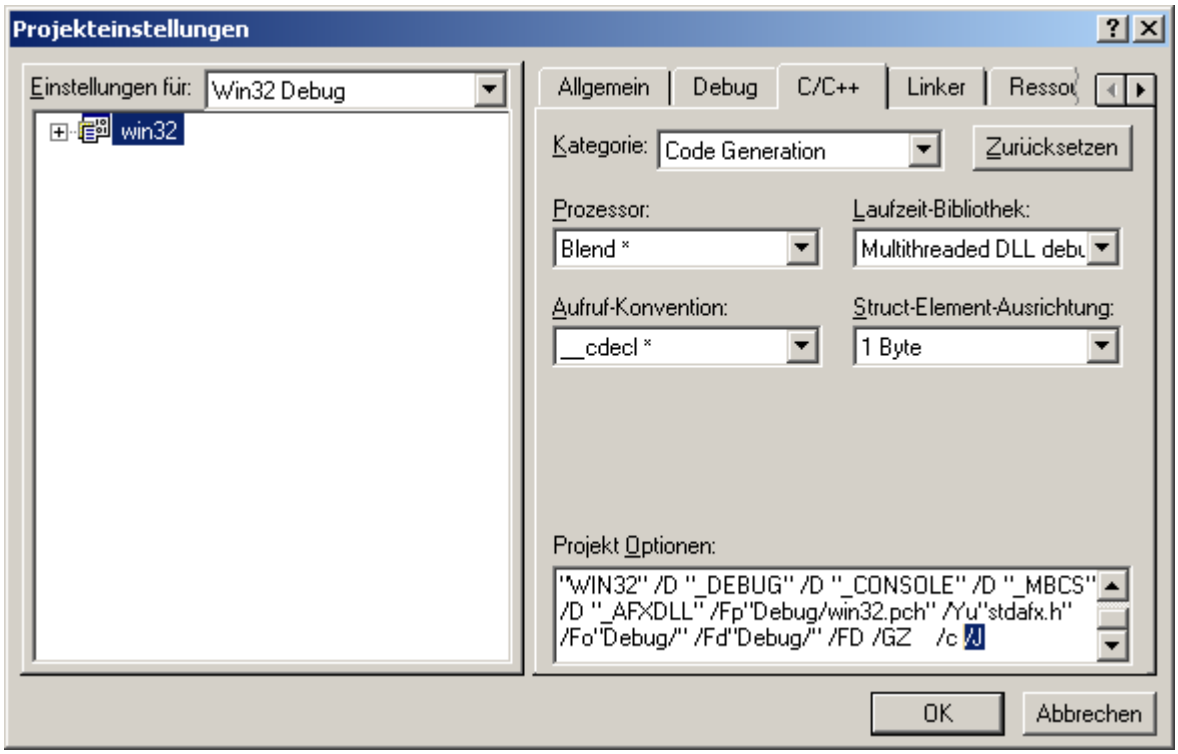
Gehen Sie nun im Menü auf „Projekt → Einstellungen“.



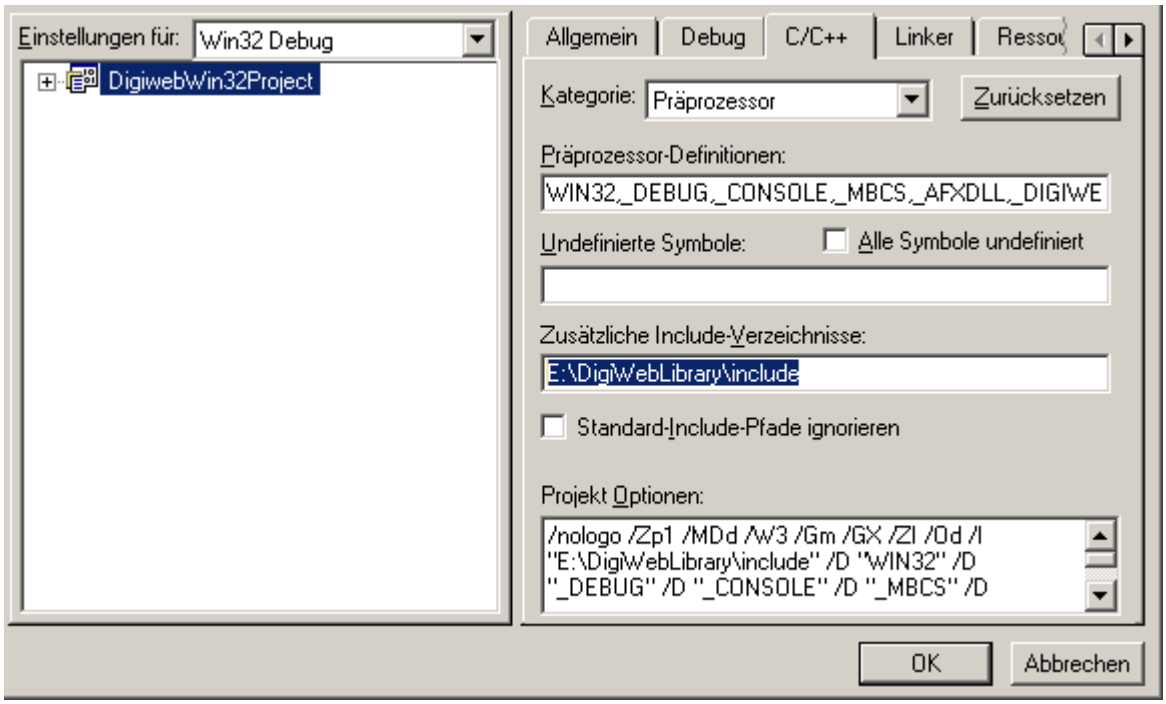
Dort wechseln Sie in den Reiter C++ und wechseln dort in die Kategorie „Code Generation“. Hier muss der Wert für „Struct-Element-Ausrichtung“ auf 1 Byte eingestellt werden.



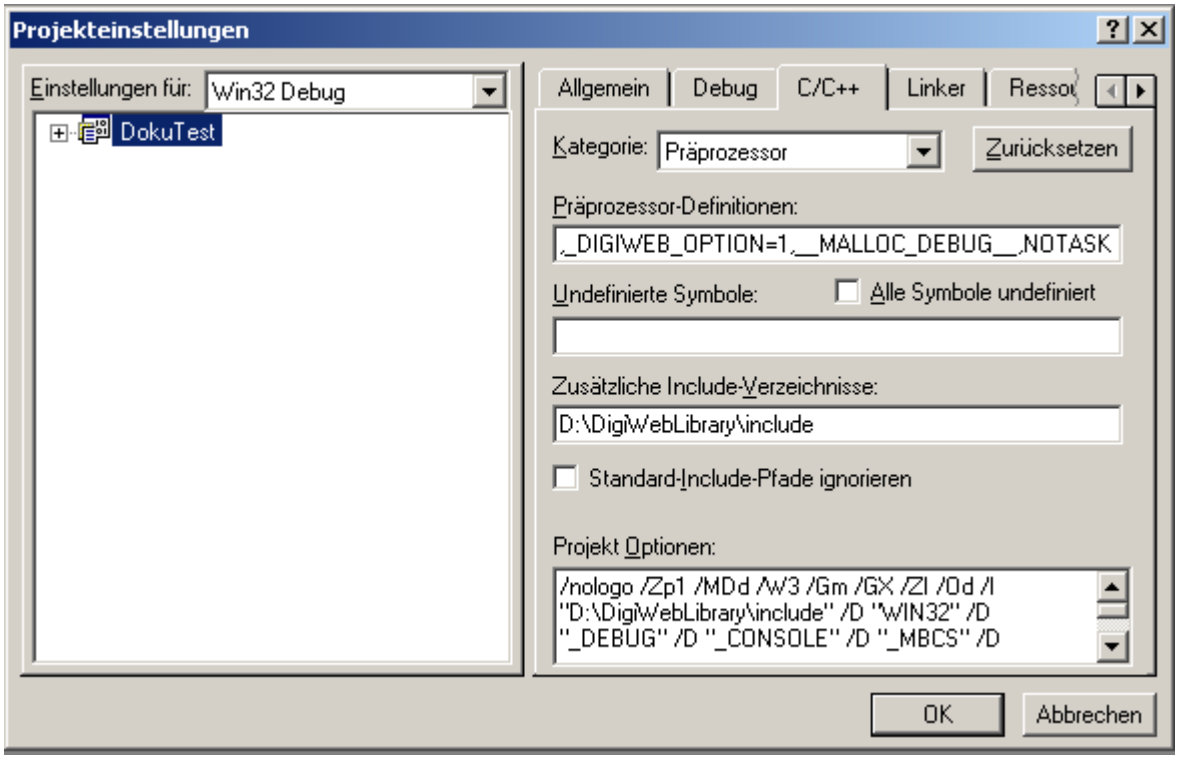
Fügen Sie den Projekt-Optionen am Ende ein „/J“ (ohne Anführungszeichen) hinzu.



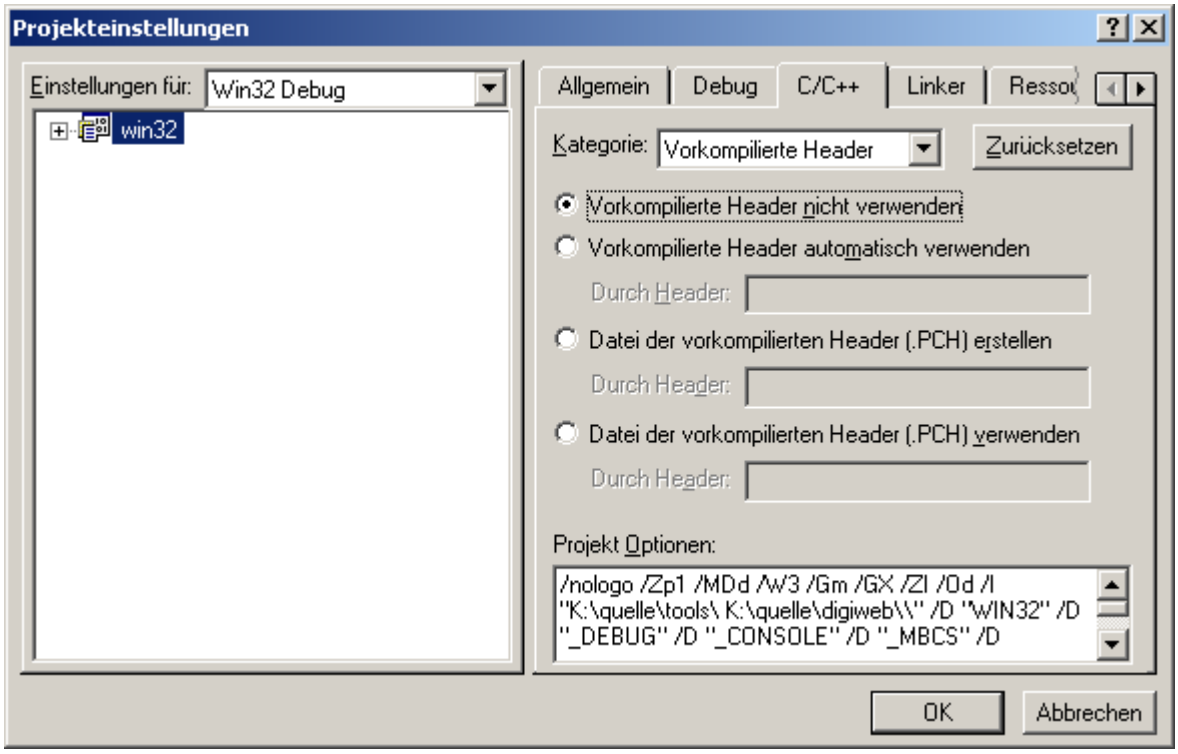
1. Wechseln Sie nun in die Kategorie Präprozessor und tragen Sie unter „Zusätzliche Include- Verzeichnisse“ folgenden Pfad ein: „*Pfad_zur_DigiWEB_Library***/include**“



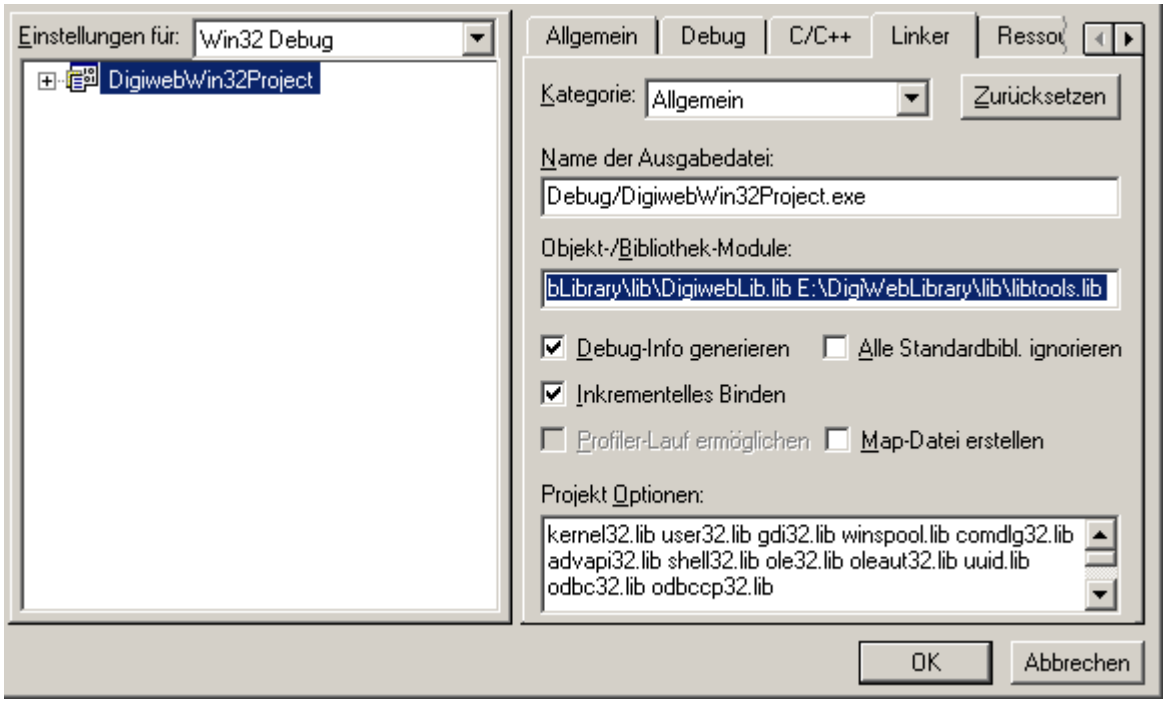
2. Fügen Sie den Präprozessor-Definitionen folgendes hinzu.
“**_DIGIWEB_OPTION=1,__MALLOC_DEBUG__,NOTASK**”



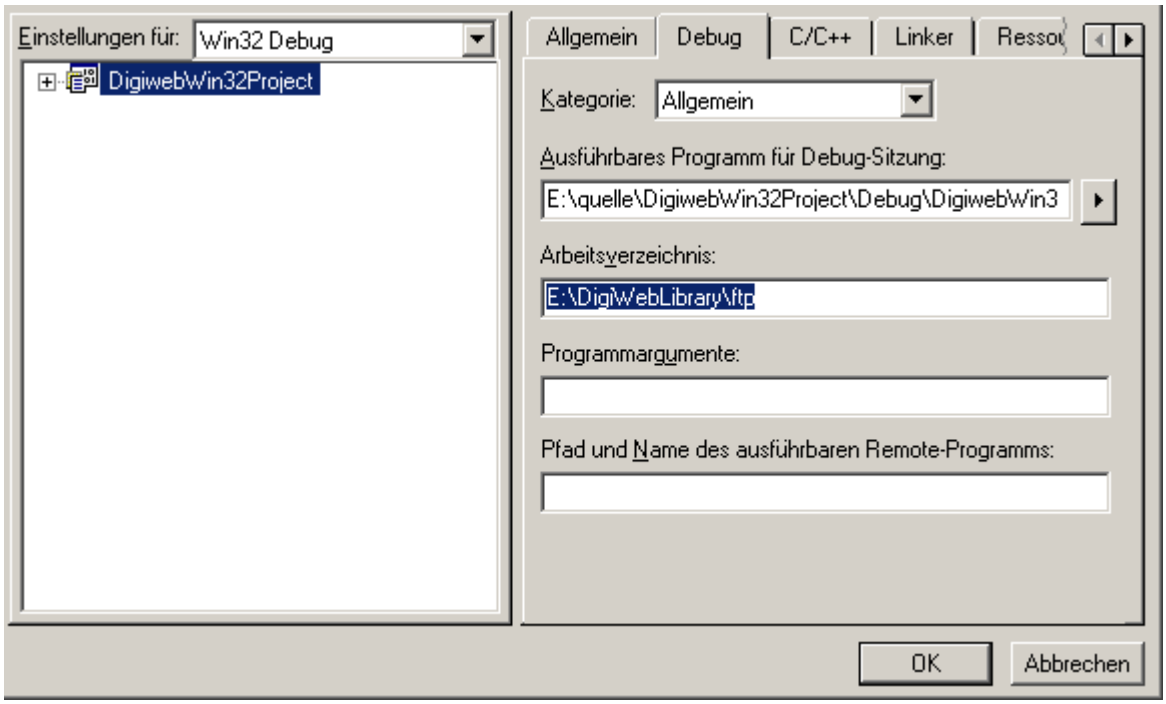
Gehen Sie zur Kategorie „Vorkompilierte Header“ und stellen Sie dort „Vorkompilierte Header nicht verwenden“ ein.



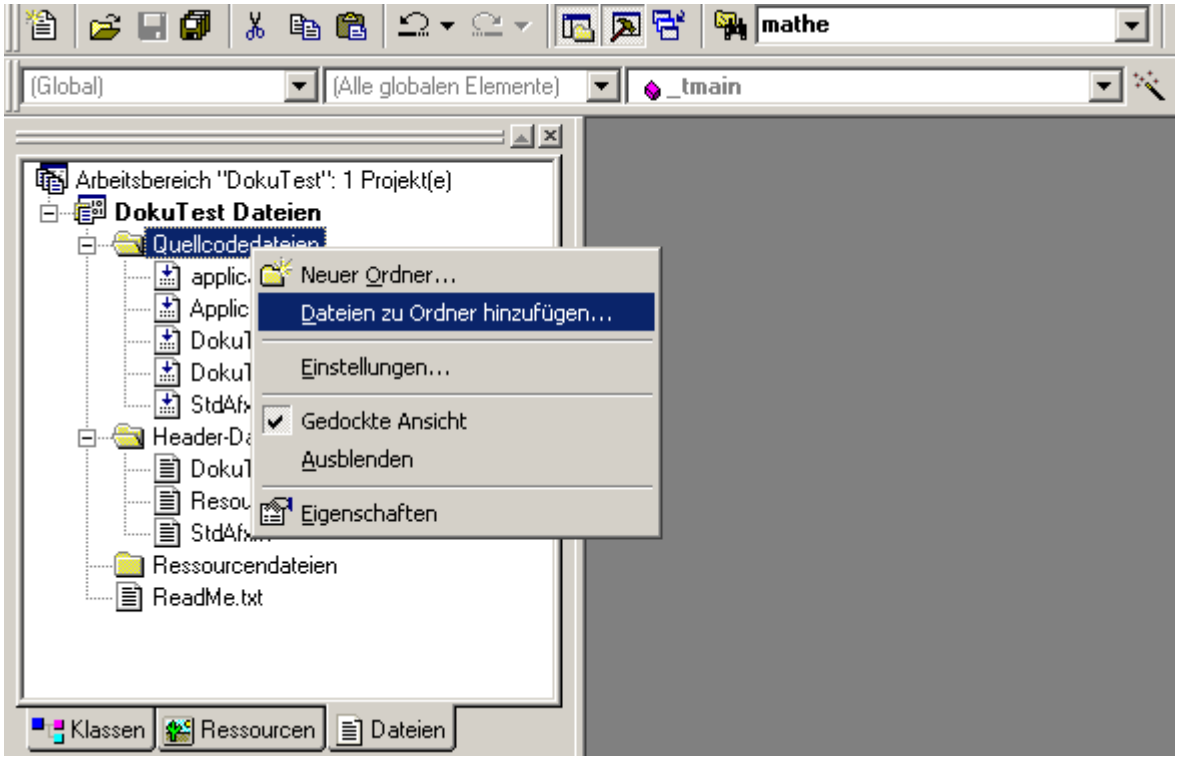
3. Wechseln Sie nun in den Reiter „Linker“. Dort tragen Sie bei „Objekt- /Bibliothek-Module“ die beiden DigiWebLibrarys ein
   -„*Pfad_zur_DigiWEB_Library*/**lib/DigiwebLib.lib**“
   -„*Pfad_zur_DigiWEB_Library*/**lib/libtools.lib**“



4. Wechseln Sie nun zum Reiter „Debug“ und tragen Sie als Arbeitsverzeichnis den Ordner „*Pfad_zur_DigiWEB_Library*/**ftp**“ ein. Anschließend klicken Sie auf OK.



5. Fügen Sie nun unter dem Reiter „Dateien“ die folgenden Dateien als Quellcodedateien zu Ihrem Projekt hinzu:
   - „*Pfad_zur_DigiWEB_Library*/**ApplicationDemo\application.c**“
   - „*Pfad_zur_DigiWEB_Library*/**ApplicationDemo\ApplicationSimu.c**“



Öffnen Sie nun das Modul, in dem die Funktion „_tmain" definiert wurde. Fügen Sie diese Funktion hinzu.

```
BOOL WINAPI _tclose(DWORD eventCode)
{
switch(eventCode)
   {
case CTRL_CLOSE_EVENT:
      MainDigiWebDestruct();
      break;
   }
   return TRUE;
}
```

Ändern Sie den Code Ihrer „_tmain" Funktion folgendermaßen ab:

```
int nRetCode = 0;
// MFC initialisieren, Ausgabe und Fehlermeldung bei Fehlern
if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(),0))
        {
// ZU ERLEDIGEN: Fehlercode gemäß Ihren Anforderungen ändern
cerr << _T("Fatal Error: MFC initialization failed") << endl;
nRetCode = 1;
        }
else
        {
SetConsoleCtrlHandler(_tclose,TRUE);
MainDigiWeb();
        }
return nRetCode;
```

Hinweis: Die Code Beispiele finden Sie in folgender Datei:
„*Pfad_zur_DigiWEB_Library***ApplicationDemo\MainModulSample.cpp**"

6. Includieren Sie im MainModul den Header „**MainDigiWEB.h**".

4 DigiWEBOptionConstruct

Diese Funktion ist der Konstruktor Ihres DigiWEB Programms. Sie wird automatisch beim Starten, nachdem die DigiWEB-Schnittstelle initialisiert wurde, aufgerufen.

```
void DigiWebOptionConstruct(void)
{
}
```

5 DigiWEBOptionDestruct

Diese Funktion wird nur für die DigiWEB Simulation am PC benötigt. Sie ist der Destruktor des DigiWEB Programms. Hier müssen alle benötigten Speicher wieder aufgelöst werden, damit die DigiWEB Simulation keine Memory-leaks im System hinterlässt. DigiWEBOptionDestruct sollte mit einer Präprozessordirektive bei allen Systemen außer WIN32 auskommentiert werden, da Sie sonst in den normalen DigiWEB-Versionen unnötigen Programmspeicher verbraucht.

```
#ifdef WIN32
void DigiWebOptionDestruct(void)
{
}
#endif
```

6 Rs232 Protokolle

Die DigiWEB-Library verlangt einen Array einer Struktur vom Typ tRsProtokoll. Hier könnten verschiedene Protokolle, die über die Rs232 Schnittstelle laufen, eingefügt werden. Standardmäßig wird einfach ein leerer Datensatz eingetragen.

```
#include „rs232connect.h“
tRsProtokoll Rs232Protokolle[]=
{
{0}
};
```

7 DigiWebOptionThread

Das DigiWEB unterstützt kein Multithreading. Deshalb müssen Funktionen, die einen längeren Zeitraum zur Abarbeitung benötigen, zwischendurch unterbrochen werden (Beispielsweise, wenn bei einer Übertragung über die serielle Schnittstelle gewartet werden muss). Diese Probleme werden im DigiWEB durch Statemachines gelöst. Es werden also Funktionen zyklisch aufgerufen, aber durch eine State Variable wird je nach Fortschritt der Abarbeitung etwas Anderes getan. DigiWebOptionThread ist eine dieser Funktionen. Sie wird in unregelmäßigen Abständen (immer, wenn genug Prozessorzeit zur Verfügung steht) aufgerufen, um komplexere Aufgaben zu erledigen. Diese Funktion darf maximal 1 Sekunde arbeiten, bis Sie die Kontrolle wieder an das System zurückgibt.

```
void DigiWebOptionThread(void)
{
}
```

8 Tickerroutinen

Tickerroutinen werden zyklisch jede in der kürzesten Zykluszeit, die das System bietet, aufgerufen. Diese Funktionen dürfen zu Ihrer Abarbeitung maximal 100 µs benötigen. Die Zykluszeit in Millisekunden steht in der Variable INT_ZYK_TIME.

8.1 Tickerroutinen anlegen

Um eine neue Tickerroutine anzulegen, wird die Funktion TickerVectorInsert() benötigt. Der Funktion muss als Parameter der Funktionspointer der neuen Tickerroutine mitgeschickt werden. Ab dem Moment wird die mitgegebene Funktion jede Millisekunde automatisch aufgerufen.

```
static void TickerInterrupt(void) //Ruft 55 mal TickerinterruptMs auf. Dadurch wird
unter Windows ein Millisekunden Interrupt simuliert
{
}

TickerVectorInsert(TickerInterrupt);
```

8.2 Tickerroutinen löschen

Um eine Tickerroutine wieder zu entfernen, muss die Funktion TickerVectorRemove() benutzt werden.

```
TickerVectorRemove(TickerInterrupt);
```

9 DIGIWEBSYMBOL_xxx

Die Symbole bilden die Schnittstelle zwischen Visualisierung und Ihrer Steuerung. Symbole können auf der Visualisierung gelesen und geschrieben werden, um so den Zustand Ihrer Steuerung zu modifizieren.

Die verschiedenen Makrotypen beginnen alle mit dem Schlüsselwort DIGIWEBSYMBOL_. Deren spezielle Eigenschaften werden durch die letzten 3 Stellen gesteuert. Die verschiedenen Einstellungen sind beliebig miteinander kombinierbar.

DIGIWEBSYMBOL_xxx Die drei ‚x' stehen jeweils für eine Einstellung.

DIGIWEBSYMBOL_**X**xx Diese Stelle steht für den Datentyp des Symbols. Auf die verschiedenen Datentypen wird später genau eingegangen.

DIGIWEBSYMBOL_x**X**x A = Symbol-Array
_ = Einfaches Symbol

DIGIWEBSYMBOL_xx**X** _ = Einfaches Symbol
R = Symbol behält nach Neustart seinen Wert (remanent)
G = Symbol kann nur gelesen werden
S = Symbol kann nur geschrieben werden
X = Substruktur mit absolutem Pointer

9.1 Die Struktur tDigiWebSymbol

Die Symbole müssen in einen Array, der vom Typ tDigiWebSymbol ist, eingetragen werden.

```
const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_L__(tAppData,SECURITY_NONE,Varname,55),
0
};
```

Der letzte Eintrag im Array muss immer {0} sein. Dieser Eintrag dient zur Endekennung. Eine fehlende Endekennung führt zu einem Absturz.

9.2 Security Areas

Jedes Symbol kann einen anderen Sicherheitslevel erhalten. Im DigiWEB werden Sicherheitslevel durch 32 Areas dargestellt. Jede Area stellt 1 Bit einer 32Bit Variable dar. Nur, wenn ein User alle Areas, die auf true stehen betreten darf, kann die Variable gelesen/geschrieben werden. Die benötigten Areas einer Variable können sich beim Lesen und Schreiben unterschreiten. Deshalb besteht jede Security Area immer aus zwei 32 Bit Variablen. Die erste Variable zum Lesen, die zweite zum Schreiben.

Konkret muss im Quellcode ein Array aus 2 long Variablen deklariert werden.
Index 0 Leseareas
Index 1 Schreibareas

```
static const unsigned long SECURITY_NONE[2] ={0x00000000,0x00000000};
```

9.3 DIGIWEBSYMBOL_L__(typ,pSecurity,name,deflong)

Dieser Symboltyp ist zum Lesen und Schreiben einer long Variable direkt in eine Struktur. Der Wert wird eins zu eins gelesen und geschrieben.

Parameter

Typ Datentyp der Struktur, in der der long deklariert wurde
pSecurity Die entsprechende Security Area
name Variablenname innerhalb der Struktur
deflong Standartwert

Beispiel:

Zugriff in Visualisierung: ##test

```
typedef struct
{
long test;
}tAppData;
tAppData ApplicationData = {0};

const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_L__(tAppData,SECURITY_NONE,test, 7),
0
};
```

DIGIWEBSYMBOL_L_R(typ,pSecurity,name,deflong) speichert den Wert remanent.

9.4 DIGIWEBSYMBOL_LA_(typ,pSecurity,name,deflong)

Dieser Symboltyp ist zum Lesen und Schreiben eines long Arrays direkt in einer Struktur. Das Symbol kann mit [] indexiert werden. Die Werte werden eins zu eins gelesen und geschrieben.

Parameter

Typ Datentyp der Struktur, in der der long deklariert wurde
pSecurity Die entsprechende Security Area
name Variablenname innerhalb der Struktur
deflong Standartwert

Beispiel:

Zugriff in Visualisierung:

##testFeld[0]
##testFeld [1]

```
typedef struct
{
long testFeld[20];
}tAppData;
tAppData ApplicationData = {0};

const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_LA_(tAppData,SECURITY_NONE,testFeld, 7),
0
};
```

DIGIWEBSYMBOL_LAR(typ,pSecurity,name,deflong) speichert die Werte Remanent

9.5 DIGIWEBSYMBOL_P__(typ, pSecurity, name,deflong)

- Wird zum Lesen und Schreiben von long Werten verwendet
- Greift nicht direkt auf den Speicher einer Variablen zu, sondern benutzt Set und Get Funktionen.

Parameter

Typ Datentyp der Struktur, in der der long deklariert wurde
pSecurity Die entsprechende Security Area
name Variablenname innerhalb der Struktur
deflong Standartwert

Wenn dieses Symbol benutzt wird, müssen die entsprechenden Set und Get Funktionen definiert werden. Das muss nach dem folgenden Schema passieren.:

long typ**Get**_name(void* pStruct)
void *typ***Set**_name(void* pStruct, long val)

Der Parameter pStruct ist die Elternstruktur des Symbols. In unserem Beispiel ist der Pointer also vom Typ AppData*.

Die Set-Funktion bekommt als Parameter den zugewiesenen Wert des Symbols als „val“.

Zum Setzen des Standardwerts wird beim Starten die Set-Funktion mit dem Wert, der bei deflong mitgeschickt wurde, aufgerufen.

Beispiel:

Zugriff in Visualisierung: ##testPrg

```
typedef struct
{
long dummy;
}tAppData;
tAppData ApplicationData = {0};

long tAppDataGet_testPrg(tAppData* pStruct)
{

return pStruct->dummy*10;
}

void tAppDataSet_testPrg(tAppData* pStruct,long val)
{
pStruct->dummy=val/10;
}


const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_P__(tAppData,SECURITY_NONE,testPrg, 1),
0
};
```

DIGIWEBSYMBOL_P_R(typ, pSecurity, name,deflong) Werte werden remanent gespeichert
DIGIWEBSYMBOL_P_G(typ, pSecurity, name) hat nur eine Get-Funktion
DIGIWEBSYMBOL_P_S(typ, pSecurity, name,deflong) hat nur eine Set-Funktion

9.6 DIGIWEBSYMBOL_PA_(typ, pSecurity, name,deflong)

- Wird zum Lesen und Schreiben von Arrays mit long Werten verwendet
- Greift nicht direkt auf den Speicher einer Variablen zu, sondern benutzt Set und Get Funktionen.
- Kann mit [] indiziert zugreifen.

Parameter

Typ	Datentyp der Struktur, in der der long deklariert wurde
pSecurity	Die entsprechende Security Area
name	Variablenname innerhalb der Struktur

long typ**Get**_name(void* pStruct, size_t index)
void *typ***Set**_name(void* pStruct, long val, size_t index)

void* pStruct Pointer auf die Elternstruktur des Symbols. In unserem Beispiel ein Pointer auf die Variable „ApplicationData“ vom Typ AppData*.
size_t index Index aus den []
long val Zugewiesener Wert in der Visualisierung

Beispiel:
Zugriff in Visualisierung:

##testPrgFeld[0]
##testPrgFeld[1]

```
long tAppDataGet_testPrg(tAppData* pStruct, size_t index)
{

return index*10;
}

void tAppDataSet_testPrg(tAppData* pStruct,long val, size_t index)
{
//hier können werte gesetzt werden
}


const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_PA_(tAppData,SECURITY_NONE,testPrg, 1),
0
};
```

DIGIWEBSYMBOL_PAR(pSecurity,target,str,array,deflong) speichert die Werte des Array remanent.
Zusätzlicher Parameter „array" ist die Maximale Anzahl an Datensätzen die remanent gespeichert werden.
DIGIWEBSYMBOL_PAS(pSecurity,target,str,deflong) hat nur eine Set-Funktion
DIGIWEBSYMBOL_PAG(pSecurity,target,str,deflong)hat nur eine Get-Funktion

9.7 DIGIWEBSYMBOL_C__(typ, pSecurity, name,defstr)

- Symbol zum Verarbeiten von Zeichenketten
- Greift direkt auf einen char* in einer Struktur zu
- Name der Variablen in der Struktur ist auch Symbolname
- Speicher wird automatisch alloziert und freigegeben

Parameter

Typ Datentyp der Struktur, in der der char* deklariert wurde
pSecurity Die entsprechende Security Area
name Variablenname innerhalb der Struktur
defstr StandardText

Beispiel:

Zugriff in Visualisierung: #$testText

```
typedef struct
{
char* testText;
}tAppData;
tAppData ApplicationData = {0};

const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_C__(tAppData, SECURITY_NONE, testText,"standard text"),
0
};
```

DIGIWEBSYMBOL_C_R(typ, pSecurity, name,defstr) speichert den Text remanent

9.8 DIGIWEBSYMBOL_CA_(typ, pSecurity, name,defstr)

- Symbol zum Verarbeiten von Zeichenketten-Feldern
- Greift direkt auf einen Array von char* in einer Struktur zu
- Name der Variablen in der Struktur ist auch Symbolname
- Zugriff wird indexiert durch []
- Speicher wird automatisch alloziert und freigegeben

Parameter

Typ Datentyp der Struktur, in der der char* deklariert wurde
pSecurity Die entsprechende Security Area
name Variablenname innerhalb der Struktur
defstr StandardText

Beispiel:

Zugriff in Visualisierung:

#$textArray[0]
#$textArray[1]

```
static struct
{
char* textArray[30];
} ApplicationData = {0};

const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_CA_(SECURITY_NONE,ApplicationData,textArray,"hallo welt"),
0
};
```

DIGIWEBSYMBOL_CAR(typ, pSecurity, name,defstr) speichert die Texte remanent

9.9 DIGIWEBSYMBOL_M__(typ, pSecurity, name,defstr,IsDestruct)

- Symbol zum verarbeiten von Zeichenketten
- Lesen des Textes über Get-Funktion
- Schreiben des Textes über Set-Funktion
- Symbolname muss keine Variable in einer Struktur erhalten
- Bei Windows zusätzliche Destruktor-Funktion möglich

Parameter

Typ Datentyp der Struktur, in der der char* deklariert wurde
pSecurity Die entsprechende Security Area
name Variablenname innerhalb der Struktur
defstr StandardText
IsDestruct 1= mit Destruktor, 0= ohne Destruktor

Funktionen

StringM* typ**Get**_*name* (tAppData* pStruct)
void typ**Set**_*name* (tAppData* pStruct, StringM* str)
void typ**Destruct**_*name* (tAppData* pStruct)

Hinweis: Der Destruktor ist nur unter Windows notwendig

Beispiel mit Destruktor:

Zugriff in Visualisierung: #$strPrg

```
typedef struct
{
long dummy;
}tAppData;
tAppData ApplicationData = {0};
static StringM *StaticText=0;

StringM* tAppDataGet_strPrg(tAppData* pStruct)
{
if(StaticText)
return StrCreateM(StaticText->str);
return StrCreateM("");
}

void tAppDataSet_strPrg(tAppData* pStruct, const char* str)
{
StaticText = StrCpyM(StaticText, str);
}

#ifdef WIN32
void tAppDataDestruct_strPrg(tAppData* pStruct)
{
if(StaticText)
        {
digiFree(StaticText);
StaticText=0;
        }
}
#endif
const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_M__(tAppData, SECURITY_NONE, strPrg,"standard text", 1),
0
};
```

DIGIWEBSYMBOL_M_R(typ, pSecurity, name,defstr,IsDestruct) speichert den Text remanent
DIGIWEBSYMBOL_M_G(typ, pSecurity, name,defstr) hat nur eine Get-Funktion

9.10 DIGIWEBSYMBOL_MA_(typ, pSecurity, name,IsDestruct)

- Symbol zum Verarbeiten von Arrays von Zeichenketten
- Lesen der Texte über Get-Funktion
- Schreiben der Texte über Set-Funktion
- Symbolname muss keine Variable in einer Struktur erhalten
- Bei Windows zusätzliche Destruktor-Funktion möglich
- Indizierung durch []

Parameter

Typ	Datentyp der Elternstruktur
pSecurity	Die entsprechende Security Area
name	Variablenname innerhalb der Struktur
defstr	StandardText
IsDestruct	1= mit Destruktor, 0= ohne Destruktor

Funktionen

StringM* typ**Get**_*name* (tAppData* pStruct, size_t index)
void typ**Set**_*name* (tAppData* pStruct, StringM* str, size_t index)
void typ**Destruct**_*name* (tAppData* pStruct)

void* pStruct Pointer auf die Elternstruktur des Symbols. In unserem Beispiel ein Pointer auf die Variable „ApplicationData“ vom Typ AppData*.
size_t index Index aus den []
StringM* str Zugewiesener Wert in der Visualisierung

Hinweis: Der Destruktor ist nur unter Windows notwendig

Beispiel mit Destruktor:

Zugriff in Visualisierung:

#$TextArray2[0]
#$TextArray2[1]

```
typedef struct
{
long dummy;
}tAppData;
tAppData ApplicationData = {0};
StringM* texte[20]={0};
StringM* tAppDataGet_strPrgFeld(tAppData* pStruct, size_t index)
{
if(index>19 || index<0)
return StrCreateM("Error out of bounds");
if(texte[index])
return StrCreateM(texte[index]->str);
return StrCreateM("str undefined");
}

void tAppDataSet_strPrgFeld(tAppData* pStruct, const char* str, size_t index)
{
if(index<20 && index>0)
texte[index]=StrCreateM(str);
}

#ifdef WIN32
void tAppDataDestruct_strPrgFeld(tAppData* pStruct)
{
int i=0;
for(i=0;i<20;i++)
        {
if(texte[i])
                {
digiFree(texte[i]);
texte[i]=0;
                }
        }
}
#endif
```

```
const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_MA_(tAppData, SECURITY_NONE, strPrgFeld, 1),
0
};
```

DIGIWEBSYMBOL_MAR(typ, pSecurity, name,array,IsDestruct) speichert die Texte remanent

DIGIWEBSYMBOL_MAG(typ, pSecurity, name) hat nur eine Get-Funktion

9.11 DIGIWEBSYMBOL_U__(typ,subtyp,name)

Dieses Symbol legt eine Unterkategorie von Symbolen an. Die Symbole werden dann mit Kategorie.Symbolname angesprochen.

Dazu muss eine neue Struktur definiert werden. Von dieser neu definierten Struktur muss dann eine Instanz in der übergeordneten Struktur angelegt werden.

Parameter:
Typ Datentyp der Elternstruktur
Subtyp Datentyp der Kindstruktur
Name Variablenname der neu angelegten Kindstruktur in der Elternstruktur

Beim Anlegen einer Substruktur wird ein neuer tDigiWebSymbol Array erwartet. Dessen Name setzt sich folgendermaßen zusammen:

*const tDigiWebSymbol subtyp**Symbols[]***
In dieser neuen Symbolstruktur können wieder alle bisher bekannten Symbolvariationen verwendet werden.

Beispiel:

Zugriff in Visualisierung: ##subData.test

```
typedef struct
{
long test;
}tUnterstruktur;

typedef struct
{
tUnterstruktur subData;
}tAppData;
tAppData ApplicationData = {0};

const tDigiWebSymbol tUnterstrukturSymbols[]=
{
DIGIWEBSYMBOL_L__(tUnterstruktur, SECURITY_NONE, test,0),
0
};

const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_U__(tAppData, tUnterstruktur, subData),
0
};
```

Jede Unterstruktur kann weitere Unterstrukturen enthalten. Die Verschachtelung ist theoretisch unendlich möglich. Das demonstriert das folgende Beispiel:

Zugriff in Visualisierung: ##Ebene2.Ebene3.test

```
typedef struct
{
long test;
}tEbene3;
typedef struct
{
tEbene3 Ebene3;
}tEbene2;
typedef struct
{
tEbene2 Ebene2;
}tEbene1;
tEbene1 Ebene1 = {0};

const tDigiWebSymbol tEbene3Symbols[]=
{
DIGIWEBSYMBOL_L__(tEbene3, SECURITY_NONE, test, 0),
0
};



const tDigiWebSymbol tEbene2Symbols[]=
{
DIGIWEBSYMBOL_U__(tEbene2, tEbene3, Ebene3),
0
};


const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_U__(tEbene1, tEbene2, Ebene2),
0
};
```

Beispiel mit mehreren Unterstrukturen eines gleichen Typs:

Zugriff in der Visualisierung:

##Subkategorie1.test
##Subkategorie2.test

```
typedef struct
{
long test;
}tEbene2;
typedef struct
{
tEbene2 Subkategorie1;
tEbene2 Subkategorie2;
}tEbene1;
tEbene1 Ebene1 = {0};

const tDigiWebSymbol tEbene2Symbols[]=
{
DIGIWEBSYMBOL_L__(tEbene2, SECURITY_NONE, test, 0),
0
};


const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_U__(tEbene1, tEbene2, Subkategorie1),
DIGIWEBSYMBOL_U__(tEbene1, tEbene2, Subkategorie2),
0
};
```

9.12 DIGIWEBSYMBOL_UA_(typ,subtyp,name)

Dieses Symbol funktioniert im Prinzip genau wie ein normales Subtarget Symbol. Der Unterschied besteht darin, dass hier ein ganzes Array von Substrukturen angelegt werden kann. Die Größe des Struktur Arrays wird automatisch erkannt. Deshalb bleiben alle Parameter und Möglichkeiten gleich.

Beispiel:
Zugriff in Visualisierung:

##Subdata[0].test
##Subdata[1].test

```
typedef struct
{
long test;
}tEbene2;
typedef struct
{
tEbene2 Subdata[20];
}tEbene1;
tEbene1 Ebene1 = {0};

const tDigiWebSymbol tEbene2Symbols[]=
{
DIGIWEBSYMBOL_L__(tEbene2, SECURITY_NONE, test, 0),
0
};


const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_UA_(tEbene1, tEbene2, Subdata),
0
};
```

10 DigiWebOptionSetGPrg

In der Visualisierung werden die Daten für Diagramme durch einen so genannten „G" Befehl abgerufen. Die Dekodierung dieses Befehls können Sie mit der Funktion DigiWebOptionSetGPrg() umleiten. Der erste Parameter muss ein Pointer auf die Haupt-tDigiWebSymbol Struktur sein. Der zweite Parameter muss die Funktion sein, die ab jetzt die „G" Befehle dekodieren soll.

```
uintcpu DigiWebG(EMUPARAMETER *emuparameter,const char *str)
{
}

void DigiWebOptionConstruct(void)
{
DigiWebOptionSetGPrg(DigiWebSymbols,DigiWebG);
}
```

Der Parameter „const char *str" enthält das Symbol, das vom Diagramm angefragt wird.
Der Parameter „EMUPARAMETER *emuparameter" ist zur Steuerung der Dekodierung notwendig.
Hier müssen die folgenden Membervariablen modifiziert werden:

emuparameter->g.opt.DiaXPrg

Programmpointer zur Funktion die die Werte der Y-Achse zurückliefert.

Diese Funktion muss folgende Parameter besitzen:

EMUPARAMETER *emuparameter Der gleiche Parameter, den auch die DigiWebG-Funktion erhält.
size_t xX Position im Diagramm

Der Rückgabewert der Funktion wird als Y-Position im Diagramm verwendet.

```
static long DigiWebGetDiagrammValue(EMUPARAMETER *emuparameter,size_t x)
{
return x*x; //Normalparabel
}
```

emuparameter->g.opt.pField

Pointer zu einem Long Array in dem die Werte der Y Achse stehen.

emuparameter->g.opt.shift

Die Werte, die „emuparameter->g.opt.DiaXPrg“ Funktion zurückliefert, werden um den hier angegeben Wert nach rechts verschoben.

L >> emuparameter->g.opt.shift

emuparameter->g.opt.bitmask

Anschließend wird das Ergebnis mit dem hier angegeben Wert UND-verknüpft.

emuparameter->g.opt.XEnd

Hier wird die Anzahl an Stützpunkten, die das Diagramm hat, eingetragen

emuparameter->g.opt.diacount

Hier muss die Anzahl der Stützpunkte als negativer Wert angegeben werden. Sinnvoll ist hier die folgende Formel:

emuparameter->g.opt.diacount=-emuparameter->g.opt.XEnd;

Rückgabewert

false = Diagramm nicht gefunden
true = Diagramm gefunden

Beispiel:

```
typedef struct
{
long Diagrammfield[50];
}tAppData;
tAppData AppData= {0};
const tDigiWebSymbol DigiWebSymbols[]=
{
DIGIWEBSYMBOL_LA_(tAppData, SECURITY_NONE, Diagrammfield, 10),
0
};


static long DigiWebGetDiagrammValue(EMUPARAMETER *emuparameter,size_t x)
{
long c=0;
long *p=emuparameter->g.opt.pField;
if((long)x<emuparameter->g.opt.XEnd)
c=p[x];
return c;
}

uintcpu DigiWebG(EMUPARAMETER *emuparameter,const char *str)
{
if(!strcmp(str, "##DiaTest[##X]"))
        {
emuparameter->g.opt.DiaXPrg=&DigiWebGetDiagrammValue;
emuparameter->g.opt.pField=AppData.Diagrammfield;
emuparameter->g.opt.shift=0;
emuparameter->g.opt.bitmask=0xffffffff;
emuparameter->g.opt.XEnd=FIELDCOUNT(AppData.Diagrammfield);
emuparameter->g.opt.diacount=-emuparameter->g.opt.XEnd;
return true;
        }
return false;
}
```

11 Eingänge lesen und Ausgänge setzen

Benötigte Header:

- Libsource/libtools.h
- QsmWegerfassung.h

11.1.1.1.1.1.1.1 Beispiele:

Hinweis! In allen Beispielen muss „n" durch die Nummer des Ausgangs/Eingangs ersetzt werden.

Ausgang einschalten: BitTableSet(QsmBufferTransmit, n);
Ausgang aussschalten: BitTableClr(QsmBufferTransmit, n);

Ausgang einlesen: BitTableGet(QsmBufferTransmit, n);
Gibt 1 zurück, wenn der Ausgang an ist; Gibt 0 zurück, wenn der Ausgang aus ist.

Eingang einlesen: BitTableGet(QsmBufferReceive, n);
Gibt 1 zurück, wenn der Eingang an ist; Gibt 0 zurück, wenn der Eingang aus ist.

12 DIGIWEBKENNUNG

Der Bootloader des DigiWEB kann verhindern, dass eine unerlaubte Software nicht in das DigiWEB eingespielt werden kann. Das ist sinnvoll, damit keine Firmware eingespielt werden kann, die nicht zur Hardware des Gerätes passt.

Um die DigiWEB Kennung zu modifizieren wird das Makro „DIGIWEBKENNUNG(Kennung) verwendet. Hier wird eine Zahl als Kennung der Software mitgegeben. Läuft das Gerät erst einmal mit einer bestimmten Kennung, verhindert der Bootloader das Installieren von Firmwares mit anderen Kennungen.

Eine Ausnahme ist die Kennung DIGIWEBKENNUNG_UNIVERSAL. Bei dieser Kennung kann jede beliebige DigiWEB Software anschließend eingespielt werden.

13 DigiWebVersion_P

Hier kann die DigiWEB Firmware Version eingetragen werden. Zum Feststellen der Version der DigiWEB Library gibt es 2 Makros.

DIGIWEBVERSION_NR	Versionsnummer der Library
DIGIWEBVERSION_DATE	Kompilierungsdatum der Library

Beispiel

```
const char DigiWebVersion_P[]=DIGIWEBVERSION_NR „Option Test („
DIGIWEBVERSION_DATE „)“;
```

14 Serielle Schnittstelle

Um im DigiWEB die Serielle Schnittstelle nutzen zu können müssen zunächst folgende Schritte befolgt werden:

1. Header inkludieren
   - rs232connect.h

2. Das Protokoll DIGIWEB_APPEND_RS_FULLDUPLEX zum RS232 Protokollarray hinzufügen. Der Protokollarray müsste bereits in Ihrem Hauptmodul vorhanden sein.

Vorher:

```
tRsProtokoll Rs232Protokolle[]=
{
   {0}
};
```

Nacher:

```
tRsProtokoll Rs232Protokolle[]=
{
   DIGIWEB_APPEND_RS_NOFRAME
   {0}
};
```

3. Im Konstruktor Ihres Projekts müssen nun die Einstellungen für die Serielle Schnittstelle gesetzt werden.

```
ComDefaultPar.u=287; //Baudrate
//mögliche Werte für Baudraten
//287  =2400        Baud
//143  =4800        Baud
//71          =9600        Baud
//35          =19200       Baud
//17          =38400       Baud
//11          =57600       Baud
//5           =115200      Baud

ComDefaultPar.u |= 0x0000;  //Bits
//mögliche Werte für Bits
//0x0000 = 8 bit
//0x0800 = 7 bit
//0x1000 = 6 bit
//0x1800 = 5 bit
```

```
ComDefaultPar.u |= 0x0000; // Parity
//mögliche Werte für Parity
//0x0000 = never
//0x2000 = even
//0x4000 = odd

ComDefaultPar.u |= 0x0000; //Stopbits
//mögliche Werte für Stopbits
//0x0000 = 1 bit
//0x8000 = 2 bit
```

4. Im Anschluss an die Konfiguration muss das Protokoll gestartet werden. Der erste Parameter spezifiziert das Protokoll. Um alle Daten ungefiltert senden und empfangen zu können muss hier " DIGIWEB_RS_FULLDUPLEX" benutzt werden. Der zweite Parameter ist die Nummer der Seriellen Schnittstelle. Für "Com1" muss hier also eine 1 eingetragen werden.

```
rs232protokoll(DIGIWEB_RS_FULLDUPLEX,1);
```

5. Daten in den Sendepuffer kopieren

```
rs232devx('a', 1); //sendet ein "a"
rs232devx(0x20, 1); //sendet ein 0x20
rs232devx('A', 1); //sendet ein 'A'
```

Beim senden ist zu beachten das Daten erstmal nur in den Sendepuffer kopiert werden, aber noch nicht wirklich gesendet werden.

6. Abfragen ob Daten im Empfangspuffer sind. Außerdem werden die Daten des Sendepuffers übertragen.

```
rs232devx(-2,1); //liefert die Anzahl der Bytes im Empfangspuffer
```

7. Daten Empfangen. Diese Funktion sollte nur genutzt werden, wenn sich Daten im Empfangspuffer befinden

```
rs232devx(-3, 1); //liefert das nächste Byte aus dem Empfangspuffer
```

15 Daten Logging

Im folgenden Kapitel wird beschrieben wird der Umgang mit Tabellen im DigiWEB beschrieben.

15.1 Tabelle erstellen

```
SQLHANDLE h = SqlConnectCsv();
SqlAppendField(h, "id", "");
SqlAppendField(h, "test", "");
SqlCreate(h, "test.csv");
SqlDisconnect(h);
```

Dieser Code würde eine Tabelle mit Namen "test.csv" anlegen. Diese würde die Spalten "id" und "test" enthalten.

```
SQLHANDLE h = SqlConnectCsv();
```

Erstellt ein SQLHANDLE, welches für alle SqlFunktionen benötigt wird. Diese Funktion allokiert Speicher. Ein SQLHANDLE wird von SqlDisconnect wieder aufgelöst.

```
SqlAppendField(h, "id", "");
SqlAppendField(h, "test", "");
```

Diese Funktion definiert die Spalten der Tabelle. Die Spalten werden in der gleichen Reihenfolge angeordnet wie die Aufrufe in C.

```
SqlCreate(h, "test.csv");
```

Erstellt eine Tabelle mit dem Namen "test.csv" und den zuvor definierten Spaltennamen. Wenn bereits eine Datei mit diesem existiert passiert nichts.

```
SqlDisconnect(h);
```

Löst den SQLHANDLE h auf und gibt dessen Speicher frei.

15.2 Datensätze anlegen

```
h = SqlConnectCsv();
SqlAppendFieldLong(h, "id", 1);
SqlAppendField(h, "test", "testext");
SqlInsert(h, "test.csv");
SqlDisconnect(h);
```

Fügt einen Datensatz zur Tabelle "test.csv" hinzu.
Mit SqlAppendField und SqlAppendFieldLong werden die Werte der einzelnen Spalten festgelegt. Sollten in der Tabelle Spalten vorhanden sein die nicht definiert wurden wird ein Leerstring eingefügt.
Die Funktion SqlInsert fügt den Datensatz in die angegebene Tabelle ein.

```
SqlInsertLog(h, "test.csv",10);
```

Diese Funktion begrenzt die Maximale Anzahl der Datensätze einer Tabelle auf den angegebenen Wert. Ansonsten Funktioniert sie genauso wie SqlInsert.

```
SqlInsertMax(h, "test.csv","id");
```

Funktioniert genau wie ein Auto-Incremental Key in MySQL der Spaltenname des Primär Schlüssels muss hier mit übergeben werden.

15.3 Verwendung von Daten aus Tabellen

```
SqlSelectAll(h, "test.csv");
```

Wählt alle Datensätze aus einer Tabelle aus.

```
SqlSelectAllWhere(h, "test.csv", "id=3");
```

Wählt alle Datensätze die zur übergebenen Where-Klausel passen.

```
SqlFetch(h, 0);
```

Setzt den Cursor auf die nächste Zeile. Liefert solange 0 zurück bis keine weiteren Datensätze mehr vorhanden sind.
Der zweite Parameter gibt die Anzahl der Zeilen an die übersprungen werden sollen.

```
SqlAccessLongByPos(h, &longVar, 1);
```

Liest einen Wert aus dem aktuell ausgewählten Datensatz.
Der zweite Parameter ist die Zielvariable.
Der dritte Parameter ist der Spaltenindex.

Weitere ähnliche Funktionen um auf die Daten zuzugreifen:

```
Intcpu SqlAccesStringByName(SQLHANDLE oi,char *dst,size_t dstlen,const char *fieldname);
intcpu SqlAccesLongByName(SQLHANDLE oi,long *dst,const char *fieldname);
StringM *SqlAccesStringByPosM(SQLHANDLE h,unsigned char pos);
StringM *SqlAccesStringByNameM(SQLHANDLE h,const char *fieldname);
```

16 DNS – Domain Namen auflösen

Die Einfache Methode:

```
tIP ip = DnsGetIp("web.de");
if(ip.is==0)
    return; //name konnte nicht aufgelöst werden
```

Da diese Funktion blockt ist sie für DigiWEB Anwendungen nicht zu empfehlen. Sie würde das gesamte DigiWEB für die Zeit in der die Anfrage läuft anhalten.

Die Methode mit einer Statemachine:

```
void (*pDnsProg)(void);
void *pvDns=0;

void waitDns(void)
{
    tIP IP;
    if((IP=DnsChk(pvDns)).is==0xFFFFffff)
    {
        return;
    }
    if(IP.is==0)
        return; //name konnte nicht aufgelöst werden
    //name wurde aufgelöst. //IP-Adresse steht nun in IP
    pvDns=0;
    pDnsProg=0;
}
void startDns(void)
{
    pvDns=DnsStart("web.de");
    pDnsProg=waitDns;
}
void DigiWebOptionThread(void)
{
    if(!pDnsProg)
        pDnsProg=startDns;
    (*pDnsProg)();
}
```

DigiWebOptionThread wird zyklisch aufgerufen.
Im ersten Schritt wird mit "void* pvDns=DnsStart("web.de");" die DNS Anfrage gestartet. Und Anschließend Schritt 2 eingeleitet.

Im zweiten Schritt wird darauf gewartet dass die Domain aufgelöst wird.
Man muss solange warten bis "DnsChk(pvDns);" etwas anderes als 0xffffFFFF zurückliefert. Wenn der zurückgelieferte gleich 0 ist, dann ist die DNS-Anfrage fehlgeschlagen. Die Domain konnte also nicht aufgelöst werden. Ist der Rückgabewert ungleich 0, dann entspricht der Rückgabewert der aufgelösten IP-Adresse.

17 Header

17.1 Libtools.h

In diesem Header sind die Schnittstellen für allgemeine Tools.

- Dateifunktionen zum Zugriff auf den internen Flashspeicher des Gerätes
- Funktionen für TCP Verbindungen
- Stringfunktionen zum Manipulieren von Strings mit eigener Speicherverwaltung
- Base 64 Kodierung
- Crc16 Kodierung
- Berechnungen mit 64 Bit Variablen
- Verkettete Listen
- Multipräzise Mathematik
- Rc4 Verschlüsselung
- Zeit-Funktionen
- Sql-Funktionen für die interne CSV Verwaltung

17.2 PWM.h

Funktionen zum Puls-weiten-moduliertem steuern von Ausgängen

17.3 Qsm.h

Funktionen zum Steuern der Ausgänge und Einlesen der Eingänge

17.4 MainDigiWeb.h

Konstruktor und Destruktor der DigiWeb Library.