Anleitung zu „h_2_bi” Version 0.2

Thomas.Freiherr@gmx.net

24. Juli 2010

Vorwort

Die vorliegende Anleitung beschreibt die Funktion und den Umgang mit dem Programm h_2_bi. Dies ist ein Tool, mit welchem Header-Dateien der Programmiersprache C in eine Header-Datei für die Programmiersprache FreeBasic übersetzt werden können. Ziel ist es, die in C geschriebenen Bibliotheken unter FreeBasic nutzbar zu machen, wie z. B. GTK+ oder sqlite3.

Zielgruppe der Software sind also Programmierer, welche die Programmiersprache FreeBasic einsetzen und deren Funktionsumfang durch den Einsatz zusätzlicher Bibliotheken erweitern wollen, durch die Aktualisierung bereits vorhandener Header oder auch durch die Generierung solcher für bisher nicht verfügbare Bibliotheken.

Dabei wird es wohl überwiegend um frei verfügbare Bibliotheken gehen. Natürlich kann h_2_bi auch für die Übersetzung von proprietären Bibliotheken verwendet werden. Die Entwickler von h_2_bi appellieren an die Anwender, die Übersetzungsergebnisse nach Möglichkeit mit anderen Anwendern zu teilen und auf dem Internet zu veröffentlichen, z. B. auf den Servern `www.freebasic.net` (en) oder `http://www.freebasic-portal.de/` (de).

Auf diesem Server befindet sich auch eine Diskussionsseite, auf welcher Erfahrungen über die Anwendung von h_2_bi ausgetauscht werden, sowie Fehler, Optimierungsvorschläge und Ideen für neue Funktionen ausgetauscht werden. Diese Diskussion wird in englischer Sprache geführt und ist unter der Adresse `http://www.freebasic.net/forum/viewtopic.php?t=15364` verfügbar.

h_2_bi hat sich in vielen Fällen im Praxiseinsatz bewährt. Dennoch können Probleme bei der Anwendung oder gar Programmfehler nicht ausgeschlossen werden, besonders wenn h_2_bi auf außergewöhnlichen Systemkonfigurationen oder in Kombination mit außergewöhnlichen Programmen eingesetzt wird. Die Nutzung von h_2_bi erfolgt auf eigene Gefahr und auf eigene Verantwortung des Anwenders.

h_2_bi ist unter Einsatz des Betriebssystems Ubuntu LINUX 9.10 entwickelt worden. Es kann auch für andere Betriebssysteme wie z. B. UNIX oder Microsoft-windows und -dos kompiliert werden.

h_2_bi ist freie Software. Das Programm wird zur öffentlichen Verwendung angeboten, in der Hoffnung dass es hilfreich ist. Es wird keinerlei Garantie und auch keinerlei rechtliche Verpflichtung übernommen, für die Folgen der Anwendung dieses Programms.

Das Programm kann weitergegeben oder modifiziert werden, unter den Bedingungen der GNU General Public License – entweder Version 3 oder wahlweise auch spätere Versionen. Einzelheiten sind unter `http://www.gnu.org/licenses/` veröffentlicht.

Inhaltsverzeichnis

1 Einleitung

h_2_bi.bas ist der Quelltext zu einem Programm (Tool), welches durch Texteingaben und Dateien von der Kommandozeile gesteuert.

Vor der ersten Anwendung muss h_2_bi.bas zunächst mit dem FreeBasic-Compiler (fbc) kompiliert werden. Das dabei erzeugte, ausführbare Programm wird im folgenden h_2_bi genannt. Natürlich wäre es möglich und für den Anwender auch bequemer, h_2_bi als anwendungsfertiges Kompilat auszuliefern, worauf hier bewußt verzichtet wird.

Bei dem späteren Einsatz von h_2_bi kann auf die Verwendung des FreeBasic-Compiler nicht verzichtet werden, sodass mit dem ersten Schritt der Selbstkompilierung sichergestellt werden kann, dass computerseitig und anwenderseitig die notwendigen Installationen und Kenntnisse für die Anwendung von h_2_bi verfügbar sind. Nach der Beschreibung von Kompilierung und Installation konzentriert sich diese Anleitung dann ganz auf die Handhabung von h_2_bi.

Das Kompilat h_2_bi ist ein Kommandozeilentool, eine grafische Benutzeroberfläche existiert derzeit nicht. Dies bedeutet, dass zum Start von h_2_biein Konsolenfenster verwendet wird. Die Handhabung ist also vergleichbar zur Anwendung des FreeBasic-Compileroder vergleichbar zur Anwendung des Programmes SWIG (-wrapper). Letzteres wird ebenso zur Übersetzung von C-Headern für FreeBasic verwendet, unterscheidet sich jedoch grundlegend in der Funktionsweise.

1.1 Dumme Übersetzung

Der SWIG-Wrapper baut auf einem C-Compiler auf. Er übersetzt den Quelltext der C-Headerdateien nicht nur in einen äquivalenten FreeBasic-Quelltext, sondern er führt – genau wie ein C-Compiler – eine Syntax-Überprüfungen durch, sowohl für die gerade bearbeitete C-Headerdatei als auch für alle eingebundenen C-Headerdateien. Wird dabei ein Fehler festgestellt, z. B. eine Datei nicht gefunden oder ein nicht definierter Variablentyp verwendet, so bricht der SWIG-Wrapper – genau wie ein C-Compiler – die Arbeit mit einer Fehlermeldung ab. Es wird kein Übersetzungsergebnis erzeugt.

Dagegen wird für die Anwendung von h_2_bi davon ausgegangen, dass die zu übersetzenden C-Headerdateien bereits erfolgreich verwendet werden. Sie enthalten also keine syntaktischen Fehler, was wohl auf die meisten publizierten Bibliotheken und die zugehörigen C-Headerdateien zutreffend sein sollte. Auch sind nicht immer alle eingebundenen C-Headerdateien notwendig, wenn diese z. B. exotische C-Compiler oder andere Rechnerarchitekturen betreffen. h_2_bi verzichtet daher auf die Syntaxüberprüfung und führt den Übersetzungsvorgang fehlertolerant durch, was in jedem Fall zu einem Ergebnis führt. Es ist dem Anwender überlassen, die Sinnhaftigkeit des Ergebnisses zu prüfen.

Das Ergebnis kann unbrauchbar sein, wenn z. B. eine wichtige C-Headerdatei nicht eingebundenen werden konnte. Im praktischen Einsatz hat sich jedoch gezeigt, dass oftmals mit wenigen Zusatzinformationen schnell brauchbare Ergebnisse erzielt werden können. Diese Zusatzinformationen werden h_2_bi mit hilfe einer Steuerdatei übergeben. Zusätzlich besteht die Möglichkeit, verschiedene Optionen für den Übersetzungsvorgang durch Kommandozeilenparameter auszuwählen.

h_2_bi erzeugt - unabhängig von der Anzahl der eingelesenen C header - für jede Steuerdatei immer genau eine Ausgabe-Datei. Diese wird im aktuellen Arbeitsordner erstellt. Sie trägt den Namen des Arbeitsordners, gefolgt von der Endung '.bi'. Sie enthält das Ergebnis der Übersetzung, ggf. C-Quelltext als Kommentar und ggf.

auch Kommentare zu Problemen, welchen beim Übersetzungsprozeß aufgetreten sind. Diese Datei wird im folgenden Text mit Ergebnisdatei bezeichnet. Der Anwender ist selbst dafür verantwortlich, ggf. Backups von dieser Datei zu erstellen, da h_2_bi diese Datei beim nächsten Aufruf ohne Vorwarnung überschreiben wird.

h_2_bi kann nicht alle möglichen C-Strukturen übersetzen, die in einem C-Header vorkommen können. Ein gewisser Teil an Handarbeit wird immer übrig bleiben. Die erfolgreiche Anwendung setzt gute Kenntnisse der C- und FreeBasic-Syntax voraus. h_2_bi wurde mit dem Ziel entwickelt, zunächst eine zuverlässige Übersetzung zu gewährleisten und dann den verbleibenden Rest möglichst gut für die manuelle Übersetzung aufzubereiten.

Als interaktives Tool wird h_2_bi für eine erfolgreiche Übersetzung üblicherweise mehrfach aufgerufen. Zwischenergebnisse werden mit dem FreeBasic-Compiler überprüft. Dazu erzeugt h_2_bi eine weitere Datei `*_Test.bas`, welche bei deren Kompilierung mit dem FreeBasic-Compilerdie Ergebnisdatei einbindet und auch für deren Kompilierung sorgt.

h_2_bi ist ein „dummer" Übersetzer, d. h. es werden z. B. keine Typ-Prüfungen vorgenommen. Diese Aufgabe erledigt später der FreeBasic-Compiler. Vorteil der „dummen" Übersetzung ist, dass in jedem Fall ein Übersetzungsergebnis vorliegt, auch wenn nicht alle verwendeten C-Typen definiert, bzw. nicht alle inkludierten Dateien installiert sind. Die Handhabung ist also gegenüber einem echten Compiler vereinfacht, jedoch müssen fehlende Informationen durch eine Steuerdatei an h_2_bi übergeben werden. Deren Inhalt bestimmt maßgeblich die Qualität des Ergebnisses.

1.2 SWIG oder h_2_bi

Der Autor hat die Arbeit an h_2_bi begonnen, weil er mit der Anwendung des SWIG-Wrappers nicht zufrieden war. Er konnte sein Ziel nicht erreichen, eine neue Version der GTK-Header für FreeBasic zu übersetzen, was dann mit einem schnell entwickelten Hilfsprogramm schließlich gelang. Aus diesem Hilfsprogramm ist durch stetige Weiterentwicklung im FreeBasic-Forum das vorliegende Programm h_2_bi entstanden.

Mittlerweile verfügt h_2_bi gegenüber SWIG über zahlreiche Zusatzfunktionen, wie z. B. :

- Der Quelltext der C-Headerdatei kann als Kommentar in die Ergebnisdatei übernommen werden.
- Es gibt vielfältige Möglichkeiten, den Übersetzungsvorgang bzw. das -ergebnis zu steuern.
- Es können Konflikte durch FreeBasic-Schlüsselworte in der C-Headerdatei vermieden werden.

2 Installation

In diesem Kapitel wird der Inhalt des Programmarchives (.zip-Datei) aufgelistet und die Schritte beschrieben, welche zur Inbetriebnahme der Software notwendig sind.

2.1 Inhalt der Zip-Datei

Die Software ist in Form einer gepackten Datei (Zip-Archiv) zum Download verfügbar. Die Zip-Datei enthält:

Tabelle 2.1: Inhalt der Datei h_2_bi_ 0.2.zip

Datei (bzw. Ordner)	Beschreibung
ReadMe.txt	Kurzanleitung in Englisch
LiesMich.txt	Kurzanleitung in Deutsch
Tutorial.pdf	Anleitung in Englisch
Anleitung.pdf	Anleitung in Deutsch (diese Datei)
h_2_bi.bas	FreeBasic Quelltext (muss noch kompiliert werden)
/bsp	Ordner mit Beispielen (Ordner für GTK+2.18.6 (inkl. leere Unterordner der C-Source) und Ordner für SQLITE-3.6.23.1 (inkl. C-Header))

Diese Datei wird zunächst auf dem Zielsystem entpackt.

2.2 Installation

Zur Anwendung der Software ist die Datei h_2_bi.bas notwendig. Diese Datei muss zunächst mit dem FreeBasic-Compiler in ein ausführbares Programm compiliert werden. Je nach verwendetem Betriebssystem ist nach der Compilierung eine Datei h_2_bi(LINUX/UNIX) oder h_2_bi.exe (windows/dos) verfügbar, welche im folgenden mit h_2_bi bezeichnet wird.

h_2_bi wurde unter Ubuntu LINUX 9.10 mit dem FreeBasic-CompilerVersion 0.20b entwickelt. Es kann auch mit anderen FreeBasic-Compiler-Versionen und für die Betriebssysteme MS-windows und -dos kompiliert werden. Jedoch wurden nicht alle möglichen Kombinationen getestet, so dass bei abweichenden Betriebssystemen oder FreeBasic-Compiler-Versionen durchaus Probleme vorkommen können.

3 Kurzanleitung

4 Bedienung

h_2_bi wurde mit dem Ziel entwickelt, C Header für die Verwendung in FreeBasic zu übersetzen. Dies geschieht bei Verwendung von standard C-Befehlen auch ohne weiteres Zutun nahezu fehlerfrei. In den meisten C Headern wird der standard Sprachumfang jedoch in mehr oder weniger großem Umfang durch Präcompiler-Macros erweitert, sodass eine Übersetzung durch h_2_bi ohne weiteres Zutun nur teilweise erfolgreich ist. Daher gibt es in h_2_bi die Möglichkeit, den Übersetzungsvorgang zu beeinflussen, indem zusätzliche Informationen bereit gestellt werden. Auch können durch die Betätigung von Schaltern verschiedene Varianten der Übersetzung eingestellt werden.

Schalter und auch Kurzinformationen werden in Form von Kommandozeilenoptionen beim Programmaufruf definiert. Ein Schalter beginnt mit einem Minuszeichen, gefolgt von einem oder mehreren Buchstaben. Es sind Groß- und Kleinbuchstaben vorgesehen, wobei Schalter mit Großbuchstaben die Eingabe eines zusätzlichen Wertes erlauben. Eine andere Form von Kurzinformationen sind Dateinamen, die in der Kommandozeile definiert werden können.

Größere Informationsmengen werden von h_2_bi aus einer Steuerdatei gelesen, wobei für jede Ergebnisdatei eine Steuerdatei bereit gestellt wird. Die Ergebisdatei erhält später den Namen der Steuerdatei. Die Steuerdatei enthält optional Informationen über den Namen des Startheaders, die Zugriffspfade für weitere Header, weitere Kommandozeilenoptionen, C-Makrodefinitionen, Ersetzungen im Basic-Quelltext und Start- bzw. Endzeilen für die Ergebnisdatei.

h_2_bi erzeugt dann während des Übersetzungsvorganges zwei Dateien, nämlich den Header für die Verwendung in FreeBasic (Endung .bi), der im folgenden Ergebnisdatei geannant wird, und eine Datei mit der Erweiterung bzw. Endung `_Test.bas`, welche zum Test der Ergebnisdatei verwendet wird.

4.1 Eingaben

4.1.1 Kommandozeile

Die Kommandozeile kann mehrere Informationen enthalten, welche durch Leerzeichen voneinander getrennt werden. Optionen im Sinne von Schaltern werden durch ein vorangestelltes Minuszeichen gekennzeichnet. Andere Zeichenfolgen werden als Dateiname interpretiert.

Die Kommandozeile wir von links nach recht nacheinander abgearbeitet. Die rechts stehenden Optionen überschreiben also ggf. zuvor spezifizierte Einstellungen. Auch können in der Steuerdatei noch weitere Optionen angegeben werden, welche nach der Kommandozeile ausgewertet werden und deren Optionen überschreiben können.

Dateinamen

Ein Dateiname wird ohne Endung angegeben. Dateinamen der Kommandozeile werden für die Steuerdatei und die Ergebnisdatei verwendet. Soll z. B. eine Ergebnisdatei mit dem Namen 'MeinHeader.bi' erzeugt werden, so ist in der Kommandozeile die Zeichenfolge 'MeinHeader' einzugeben. h_2_bi sucht dann nach der Steuerdatei 'MeinHeader.h2bi', aus welcher die Infromationen für den Übersetzungsvorgang gelesen werden. Nach erfolgter Übersetzung wird die Ergebisdatei 'MeinHeader.bi' erzeugt.

Die Kommandozeile kann auch mehrere Dateinamen enthalten. Für jeden Dateinamen wird eine Ergebnisdatei erzeugt, wenn die entsprechende Steuerdatei vorhanden ist. Wird ein Dateiname angegeben, für den keine Steuerdatei existiert, so kann h_2_bi den Übersetzungsvorgang nicht starten. Stattdessen wird dem Anwender angeboten, eine neue, leere Steuerdatei mit diesem Namen zu generieren.

Ist kein Dateiname in der Kommandozeile angegeben, so versucht h_2_bi Standard-Dateinamen anzuwenden. Zunächst wird nach einer Steuerdatei mit dem Namen des aktuellen Ordners gesucht. Existiert diese nicht, so wird nach der Datei 'h_2_bi.h2bi' gesucht. Falls eine dieser Dateien gefunden wird, so wird eine Ergebnisdatei mit dem gleichen Namen und der Endung '.bi' erzeugt. Andernfalls wird h_2_bi beendet.

Kommandozeilenoptionen

Kommandozeilenoptionen beginnen mit einem Minuszeichen. Die jeweilige Option wird durch einen Groß- oder Kleinbuchstaben spezifiziert. Es können mehrere Optionen nacheinander angegeben werden. Nach einem Leerzeichen muss die Spezifikation weiterer Optionen erneut mit einem Minuszeichen eingeleitet werden.

Großbuchstaben-Optionen können mit einem ganzzahligen Wert ergänzt werden, der dezimal, binär, oktal oder hexadezimal angegeben wird. Die Interpretation des Wertes erfolgt entsprechend der Arbeitsweise des FreeBasic VALINT Befehles. Fehlt die Angabe eines Wertes, so wird dies als Wert 0 (Null) interpretiert.

C Mit der Option `-C` wird die Interpretation der Steuerdatei beeinflußt. Der folgende Wert beinhaltet 32 einzelne Schalter, mit welchen die Verwendung einzelner Sektionen der Steuerdatei abgeschaltet werden kann. Die Funktionen der einzelnen Schalter sind in Tabelle 8.5 detailiert beschrieben. Wird kein Wert angegeben, so werden alle Schalter ausgeschaltet.

O Mit der Option `-O` wird die Erstellung des Basic-Quelltextes beeinflußt. Der folgende Wert beinhaltet 32 einzelne Schalter, welche Einfluß auf die Gestaltung der Ergebnisdatei haben, und zwar auf den FreeBasic-Quelltext, also jenen Text-Anteil, der vom FreeBasic-Compilerspäter verwendet wird. Die Funktionen der einzelnen Schalter sind in Tabelle 8.7 detailiert beschrieben. Wird kein Wert angegeben, so werden alle Schalter ausgeschaltet.

I Mit der Option `-I` wird die Erstellung der Kommentare in der Ergebnisdatei beeinflußt. Der folgende Wert beinhaltet wiederum mehrere Schalter, welche Einfluß auf die Gestaltung des C-Quelltextes in der Ergebisdatei haben, welcher optional aus dem C-Header als Kommentar in die Ergebnisdatei übernommen werden kann. Wird kein Wert angegeben, so werden alle Schalter ausgeschaltet und kein Quelltext aus dem C-Header übernommen.

D Mit der D-Option wird ein sogenannter Debug-Level spezifiziert. Ein solcher Level dient der Bequemlichkeit und beeinflußt mehrere Schalter der B- und der C-Option gleichzeitig. Die einzelnen Debug-Level sind in den Tabellen 8.5, 8.6 und 8.7 detailiert beschrieben. Es kann der gleiche Level für die B- und C-Schalter spezifiziert werden, indem der Wert durch eine einzelne (Dezimal-) Ziffer angegeben wird. Bei zweistelligen Dezimal-Werten wird der Level der C-Option entsprechend der ersten Ziffer und der Level der B-Option entsprechend der zweiten eingestellt. Wird kein Wert angegeben, so werden alle B- und C-Schalter ausgeschaltet.

S Mit der S-Option wird die Art der Einrückungen in der Ergebnisdatei beeinflußt. Standardmäßig wird für eine Einrückung ein Tabulatorzeichen (CHR(9)) verwendet. Es ist auch möglich, Leerzeichen für Einrückungen zu verwenden. Dazu wird mit dem Wert der S-Option die Anzahl der Leerzeichen spezifiziert. Wird der Wert 0 (NULL) angegeben (bzw. kein Wert eingegeben), so werden die Einrückungen für die Ergebnisdatei abgeschaltet, also gar keine Einrückungen erzeugt.

Kleinbuchstaben-Optionen dienen überwiegend der Bequemlichkeit, da einzelne Buchstaben einfacher zu merken sind als die entsprechenden Schalter der B-, C- oder D-Optionen.

n Die n-Option schaltet die Ausgabe von Variablennamen in Parameterlisten ein. FreeBasic benötigt in der Parameterliste einer Prozedur (SUB) oder Funktion lediglich die Angabe der Variablentypes. Für die menschliche Interpretation der Definition kann es sinnvoll sein, zusätzlich den Namen der Variablen anzugeben, was durch die Spezifikation dieser Option erreicht wird.

w Die w-Option schaltet den Zeilenumbruch (word wrap) für Parameterlisten von Prozeduren (SUB) oder Funktionen ein. Diese können bei Verwendung mehrerer Parameter lang und unübersichtlich werden. Bei Verwendung des Wortumbruches wird jeder Parameter jeweils in eine Zeile geschrieben, welche dann durch einen Unterstrich (_) für den fbc mit der nächsten Zeile verbunden wird.

v Die v-Option erzeugt die Ausgabe von Versionsinformationen zum h_2_bi-Programm. Sie bewirkt die Beendung des Programmes, nachdem eine Zeile mit der Versionsnummer und dem Zeitstempel der Kopilierung (Datum und Uhrzeit), sowie der Plattform (LINUX oder MS-dos/win) ausgegeben wird.

h Die h-Option dient zur Ausgabe eines Hilfetextes. Sie bewirkt die Beendung des Programmes, nachdem mehrere Zeile mit Hinweisen über die Programmbenutzung und Erklärungen zu den Parametern ausgegeben werden.

f Die f-Option dient zur Erzeugung einer „schnellen" Ergebnisdatei. Sie setzt alle Schalter der B- und C-Optionen so, dass der Quelltext der Ergebnisdatei möglichst schnell vom FreeBasic-Compiler verarbeitet werden kann. Z. B. wird C-Quelltext nur übernommen, wenn ein Übersetzungsproblem festgestellt wird. Parameterlisten werden nicht umgebrochen, sie enthalten keine Namen, usw. Die detailierte Schaltereinstellungen können den Tabellen 8.5, 8.6 und 8.7 entnommen werden.

t Die t-Option dient der Abschaltung der Übersetzung der C-Variablentypen, welche in h_2_bi vordefiniert sind, vgl. Tabelle 8.3 auf Seite 34. Die Übersetzung dieser Standardtypen kann mit der t-Option abgeschaltet werden, wodurch zunächst ein unbrauchbares Übersetzungsergebnis entsteht. Eine alternative Typenumwandlung kann dann durch entsprechende Makrodefinition definiert werden.

4.1.2 Steuerdatei

Als Steuerdatei wird von h_2_bi eine Textdatei mit der Endung '.h2bi' gelesen. Diese wird entweder von h_2_bi generiert oder mit einem beliebigen Texteditor erzeugt und bearbeitet. Diese Datei wird im Arbeitsordner gesucht, also in jenem Verzeichnis, aus welchem der Aufruf von h_2_bi erfolgt.

Die Steuerdatei ist durch Überschriften in mehrere Sektionen unterteilt. Die Zeilen innerhalb der Sektionen haben jeweils eigene Funktionen, bzw. wirken an unterschiedlichen Stelle auf den Übersetzungsvorgang. SektionsÜberschriften sind notwendig, um die folgenden Zeilen der entsprechenden Funktion zuzuweisen.

Die Reihenfolge der Sektionsüberschrift(en) ist beliebig, Überschriften können auch mehrfach definiert werden. Wenn eine oder mehrere Sektionen nicht benötigt werden, dann kann auf die entsprechende(n) Überschrift(en) auch verzichtet werden.

Sektion `__PARAMETER__(){};`

Diese Sektion dient zur Eingabe zusätzlicher Optionen, ergänzend zu jenen der Kommandozeile. Es können mehrere Optionen in einer Zeile definiert werden. Oder es wird jede weitere Option in eineZeile geschrieben. welche dann zusätzlich mit einem Kommentar ergänzt werden kann.

Die Optionen dieser Sektion werden nach der Kommandozeile ausgewertet und überschreiben ggf. dortige Definitionen.

Sektion `__PATHS__(){};`

Diese Sektion dient zur Definition von Zugriffspfaden, in welchen h_2_bi nach C-Headern suchen soll. Sowohl der Startheader als auch weitere Headerdateien, welchen durch den Startheader eingebunden werden, müssen nicht im Arbeitsverzeichnis existieren, sondern sind gewöhnlich in teilweise verschachtelten Unterverzeichnissen enthalten.

Jede Zeile dieser Sektion enthält jeweils einen Zugriffspfad, welcher von h_2_bi benutzt wird, um die Startdatei oder weitere, mit dem C-Befehl **`#include ...`** eingefügte C-Header aufzufinden. Die Zugriffspfade werden in der Reihenfolge ihrer Definition abgesucht, h_2_bi verwendet die zuerst gefundene Datei.

Sektion `__BI_REPS__(){};`

Diese Sektion dient zur Definition von Basic-Headern, welche bereits übersetzt wurden und in die Ergebnisdatei eingebunden werden sollen. Viele C-Header binden wiederum andere C-Header ein, welche teilweise zu der Bibliothek gehören, teilweise aber auch aus anderen Bibliotheken wie z. B. der C-Standardbibliothek stammen können. FreeBasic enthält bereits Header für die C-Standardbibliothek, welche von h_2_bi automatisch verwendet werden, falls diese Funktion nicht abgeschaltet wird.

Die Liste der bereits übersetzten Header kann durch Einträge in dieser Sektion ergänzt werden. In einer Zeile kann jeweils ein Header definiert werden, indem zunächst der Name des C-Headers angegeben wird, gefolgt von dem Zeichen '>'. Danach wird der Zugriffspfad und der Name des FreeBasic-Headers definiert. Eine entsprechende Zeile wäre z. B.

```
MeinHeader.h>crt/MeinHeader.bi
```

Der Zugriffspfad ist entweder absolut anzugeben, oder er kann (bevorzugt) relativ zu dem 'inc'-Pfad der FreeBasicInstallation angegeben werden (= der Pfad, in welchem der FreeBasic-Compiler nach '.bi'-Dateien sucht). Wenn h_2_bi in der C-Headerdatei dann auf eine Zeile **`#include`** `<MeinHeader.h>` trifft, wird diese zu `#`**`INCLUDE ONCE`** `"crt/MeinHeader.bi"` übersetzt. Dies gilt auch, wenn die Zeile im C-Header **`#include`** `"MeinHeader.h"` lautet.

Sektion `__START_BI__(){};`

Diese Sektion dient zur Ergänzung der Ergebnisdatei. Die Zeilen dieser Sektion werden ohne Veränderung vor das Übersetzungsergebnis in die Ergebnisdatei eingefügt, inklusiv der Leerzeilen.

Die Zeilen dieser Sektion enthalten also FreeBasic Quelltext oder Kommentare. Wenn keine solche Sektionsüberschrift in der Steuerdatei existiert, dann beginnt die Ergebnisdatei direkt mit dem Übersetzungsergebnis.

Diese Sektion Datei kann z. B. dazu genutzt werden, Copyright Informationen in die Ergebnisdatei zu schreiben. Oder es können FB-Befehle ergänzt werden, welche nicht während der Übersetzung generiert werden, wie z. B. die Befehle für die Einbindung der Binärdateien einer Bibliothek (`#`**`inclib`** `"␣...␣"`). Oder es können Befehle für Typ- oder Makro-Definitionen eingefügt werden.

Oder es kann ein Befehlsblock (oder mehrere) eröffnet werden, welcher das gesamte Übersetzungsergebnis von h_2_bi umschließt. In diesem Fall muss der Befehlsblock in der Sektion `__END_BI__(){};` abgeschlossen werden, damit eine fehlerfreie Kompilierung mit dem fbc möglich wird. Z. B. kann diese Sektion den Befehl `#IFNDEF __MeinHeader_BI__` enthalten und der zugehörige Befehl `#`**`endif`** ist in der Sektion `__END_BI__(){};` enthalten.

Sektion `__END_BI__(){};`

Diese Sektion dient ebenfalls zur Ergänzung der Ergebnisdatei. Ihre Zeilen werden unverändert hinter das Übersetzungsergebnis in die Ergebnisdatei eingefügt. Sie enthalten FreeBasic Quelltext oder Kommentare.

Wenn keine solche Sektionsüberschrift in der Steuerdatei existiert, enthält die Ergebnisdatei nach dem Übersetzungsergebnis direkt das Protokoll, sofern dessen Erzeugung nicht abgeschaltet ist.

Primärer Sinn dieser Sektion ist der Abschluß von Befehlsblöcken, welche in der Sektion `__START_BI__(){};` eröffnet wurden. Natürlich können nach Belieben auch weitere FB Befehle eingefügt werden oder die Ergebnisdatei mit abschließenden Kommentare versehen werden.

Sektion `__MACROS__(){};`

Diese Sektion dient zur Definition von Makros, welche beim Übersetzungsvorgang expandiert werden sollen. Als 'dummes' Übersetzungsprogramm führt h_2_bi bei Makros normalerweise nur geringe Syntaxanpassungen durch und übernimmt sie ansonsten unverändert in die Ergebnisdatei. Eine weitere Berücksichtigung während der Übersetzung findet nicht statt.

Manche C-Makros müssen jedoch berücksichtigt werden, um eine lauffähige übersetzung zu erzeugen. Teilweise müssen diese entfernt werden, da sie für FreeBasic keine Bedeutung haben, was durch Definition eines leeren Makros erreicht wird. Teilweise müssen Makros aber auch expandiert werden, weil erst durch diesen Vorgang der zu übersetzende C-Quelltext vollständig wird.

Die Definition solcher Makros erfolgt in dieser Sektion. Die Zeilen entsprechen syntaktisch der Makrodefinition im C-Header. In den meisten Fälle genügt es, die benötigte Makrodefinition aus dem C-Header in diese Sektion zu kopieren. Natürlich können im Makrotext auch Anpassungen vorgenommen werden, um ein für FreeBasic günstiges Ergebnis zu generieren. Solche Anpassungen müssen natürlich wieder der C-Syntax entsprechen.

Die Expansion der Makros erfolgt vor dem Übersetzungsvorgang. Wenn ein Makrotext ein weiteres Makro enthält,dann wird auch das innere Makro vor der Übersetzung expandiert. Diesen Vorgang nennt man rekursive Expansion. Sie kann zu Endlosschleifen führen (das Programm 'hängt'), wenn ein Makro sich selbst enthält, oder - weniger leicht erkennbar - innerhalb von mehreren verschachtelten Makros das erste Makro enthalten ist.

Sektion `__BASIC_REPS__(){};`

Diese Sektion wirkt auf FreeBasic-Quelltext nach dem Übersetzungsprozeß. Es werden Ersetzungen bzw. Ergänzungen von Worten definiert. Dieser Vorgang ist ähnlich der Funktion 'Suchen und Ersetzen' in einem Editor. Die Suche erfolgt unter Berücksichtigung der Groß- und Kleinschreibung. Gefunden werden jedoch nur ganze Worte im FreeBasic-Quelltext (nicht im C Quelltext), bei Suchtexten mit Leerzeichen oder Rechenzeichen wird nur das erste Wort berücksichtigt.

Worte bestehen aus jenen Zeichen, welche für die definition von Variablennamen zulässig sind, also '`a`' bis '`z`', '`A`' bis '`Z`', '`0`' bis '`9`' und der Unterstrich '_'. Ein Wort darf nicht mit einer Ziffer oder einem Unterstrich beginnen.

Die Definition einer Suchfunktion erfolgt jeweils in einer Zeile, welche zunächst das Suchwort enthält, gefolgt von einem Steuerzeichen '>' oder '&'. Wenn kein Steuerzeichendefiniert ist, wird die Zeile ignoriert.

Das Steuerzeichen '`>`' bedeutet, dass das Suchtwort durch die folgende Zeichenkette ersetzt wird. Die Zeichenkette bzw. der restliche Text der Zeile wird anstelle des Suchwortes in den FreeBasic-Quelltext eingefügt.

Das Steuerzeichen '`&`' bewirkt, dass das Suchwort im FreeBasic-Quelltext durch die folgende Zeichenkette ergänzt wird (die Zeichenkette wird an das Suchwort angehängt). Wenn keine Ergänzungszeichenkette definiert ist, wird die Standardergänzung `_TJF` verwendet.

Diese Funktion wurde ursprünglich implementiert, um Groß- und Kleinschreibung von C Makro-, Variablen- oder Typnamen in eine geeignete FB Form zu bringen. Wenn z. B. die C-Headerdatei eine Funktion mit dem Namen `aaa` und ein Makro namens `AAA` enthält und diese ohne weiteres Zutun übersetzt wird, dann wird der FreeBasic-Compiler die Kompilierung mit der Fehlermeldung `duplicated definition` abbrechen. Durch eine Zeile `AAA&` in der Steuerdatei

wird das Makro im FreeBasic-Quelltext in `AAA_TJF` umbenannt und es kann eine fehlerfreie Kompilierung erfolgen.

Diese Funktion kann auch dazu genutzt werden, um die Schreibweise der FB Befehle an die eigenen Wünsche anzupassen. Bei der Entwicklung von h_2_bi hat es sich bewährt, alle FreeBasic-Befehle in großen Buchstaben zu schreiben. Dadurch ist eine einfache Unterscheidung möglich, ob ein Befehl unverändert aus der C-Headerdatei übernommen wurde, oder ob das Wort durch h_2_bi übersetzt in die Ergebnisdatei gelangte. Zur Veränderung der Schreibweise wird als Suchwort der Befehl in Großbuchstaben eingegeben und als Ersetzung derselbe Befehl in der gewünschten Schreibweise, z. B. ändert die Zeile **`TYPE>Type`** die Schreibweise aller TYPE-Befehle.

4.2 Ausgaben

h_2_bi erzeugt Ausgaben auf zwei verschiedene Arten. Einerseits werden Meldungen in derjenigen Konsole ausgegeben, von welcher der Programmaufruf erfolgte. Andererseits werden Ausgaben in Textdateien geschrieben, welche von h_2_bi im Arbeitsordner generiert werden, sofern der Anwender über Schreibrechte in diesem Ordner verfügt.

Im Arbeitsordner werden zwei Dateien generiert. Einerseits erzeugt h_2_bi bei fehlerfreiem Lauf eine neue Ergebnisdatei, welche die Übersetzung des oder der C-Header enthält. Und andererseits wird ggf. eine Testdatei mit FreeBasic-Quelltext erzeugt, sofern diese noch nicht existiert. Letztere dient zur Überprüfung der Ergebnisdatei mit dem FreeBasic-Compiler.

4.2.1 Textausgabe im Konsolenfenster

Textausgaben im Konsolenfenster geben dem Anwender Informationen über den aktuellen Bearbeitungsstand des h_2_bi-Aufrufes. Der Beginn jedes wesentlichen Bearbeitungsschrittes wird mit einer Meldung angezeigt. Treten in einem Bearbeitungsschritt Fehler auf, so meldet h_2_bi dies vor der Beendung des Programmes. Die Meldungen der Konsole dienen also der Überwachung und Optimierung des Programmablaufes von h_2_bi.

4.2.2 Ergebnisdatei

Die Erzeugung dieser Datei ist das primäre Ziel der Anwendung von h_2_bi. Sie enthält das Ergebnis der Übersetzung des/der C Header, also den FreeBasic-Quelltest und gegebenenfalls auch weitere Informationen, sofern deren Generierung durch die Parametervogabe nicht unterbunden wird.

Der Name dieser Datei ist nicht fest vorgegeben, sondern die Datei wird nach dem Namen des Arbeitsordners benannt, ergänzt durch die Endung .bi.

Der Aufbau der Ergebnisdatei ist in vier Blöcke unterteilt:

1. Inhalt der Datei Start.bi
2. Ergebnis der Übersetzung der C Header
3. Inhalt der Datei Ende.bi
4. Protokoll

Durch Verwendung entsprechender Kommandozeilenparameter bzw. -optionen können die Informationsblöcke 1, 3 und 4 abgeschaltet werden.

Der Inhalt der Blöcke der .bi-Dateien (1 und 3) wird durch den Inhalt der entsprechenden Steuerdateien (Start.bi und Ende.bi) bestimmt. h_2_bi übernimmt diese Dateien unverändert in die Ergebnisdatei.

Der Block 2 enthält das Ergebnis des Übersetzungsprozeß. Hier findet sich der von h_2_bi generierte FreeBasic-Quelltext, sowie ggf. Meldungen, welche h_2_bi beim übersetzen generiert.

Im letzten Block findet sich die Protokollsection, welche informiert über:

- den Zeitpunkt der Übersetzung,

- die Parameter beim Aufruf von h_2_bi,
- die verwendeten Typumwandlungen (types.rep)
- die verwendeten Makro-Namen (macro.rep)
- die verwendeten Wortersetzungen (post.rep),

Meldungen in der Ergebnisdatei

Wenn während des Übersetzungsprozesses der Quelltext des C Headers nicht den Erwartungen entspricht, markiert h_2_bi die entsprechende Stelle mit einem Kommentar im FreeBasic-Quelltext. Dieser enthält drei Fragezeichen und einen Hinweis, welche Erwartung nicht erfüllt wird. In diesem Kapitel sind die möglichen Meldungen von h_2_bi und geeignete Abhilfemaßnahmen einzeln beschrieben.

xxfile . . .

. . .

Meldungen, welche mit einer zweistelligen Zahl gefolgt von dem Wort file (= englisch Datei) beginnen,

4.2.3 Testdatei ('*_Test.bas')

Diese Datei dient zum Test der Ergebnisdatei, h_2_bi erzeugt sie aus Bequemlichkeitsgründen. Sie ist benannt nach dem Namen der Steuerdatei und wird erzeugt, wenn h_2_bi erstmalig mit einer Steuerdatei aufgerufen wird. Existiert bereits eine entsprechende Datei, so verändert bzw. überschreibt h_2_bidiese nicht.

Diese Datei enthält zunächst nur eine einzige Zeile mit einem `#INCLUDE`-Befehl, welcher die Ergebnisdatei (den neu erzeugten FreeBasic-Header) einbindet. Durch die Compilierung dieser Datei mit dem FreeBasic-Compilerwird also die Ergebnisdatei übersetzt. Der FreeBasic-Compilermeldet bei diesem Vorgang ggf. Fehler, welche in der Ergebnisdatei enthalten sein können.

Diese Datei kann für weitergehende Testzwecke durch FreeBasic-Quelltext ergänzt werden, z. B. um einzelne Funktionen der Ergebnisdatei zu testen.

5 Funktionen

Im vorangegangenen Kapitel ist die Bedienung von h_2_bi beschreiben. Es ist beschrieben, wie einzelne Funktionen ein- bzw. ausgeschaltet werden können. In diesem Kapitel sind diese Funktionen detailiert beschrieben, wobei zunächst auf die Textausgabe der Konsole eingegangen wird und anschließend die Inhalte der Ergebnisdatei beschrieben wird.

5.1 Konsolen-Textausgabe

5.2

5.3

5.4

5.5

5.6 Basic Quelltext (Ergebnisdatei)

Schalter steuern Funktion

Drei Schaltergruppen C, I, O - Bedeutung

Bedienung der Schalter entweder durch Optionen -c, -i, -o oder mit -p

Dieser Wert besteht in seiner Binärform aus einer Kette von 32 Nullen und Einsen. Jedes dieser Zeichen (Bit) stellt einen Schalter für die entsprechende Funktion dar. Eine Funktion wird eingeschaltet, indem das entsprechende Bit gesetzt wird (Wert = 1). Mit dem Wert hinter der Option -B werden also alle Schalter gleichzeitig gesetzt. So schaltet z. B. die Angabe `-B0` alle Funktionen aus.

Um nun den benötigten Wert für die Option `-B` zu ermitteln, können entweder die Dezimalwerte der Bits (Schalter) aufaddiert werden und die so ermittelte Dezimalzahl hinter der Optionskennung `-B` angegeben werden. Alternativ kann der Wert auch als Binärzahl angegeben werden, wobei jedes einzuschaltende Bit durch eine 1 repräsentiert wird. Eine binäre Zahl wird durch die Zeichenfolge `&b` eingeleitet, wie z. B. `&b1101`.

5.6.1 Gruppe C

Die Kennung `C` steht für „Config file", mit dieser Option werden die Funktionen zur Interpretation der Steuerdatei geschaltet. Die Funktionen können entweder einzeln durch die Verwendung der Option `-P` geschaltet werden, oder alle Schalter werden gleichzeitig mit der Option `-C` bedient, vgl. Tabelle 5.6.1 auf Seite 16.

Die Schalter haben folgende Bedeutung:

5.6.2 TY

Die Abkürzung dieses Schalters steht für **C**onfig file **TY**pe-Declarations. Mit ihm wird die Verwendung der Sektion `__TYPES__(){};` in der Steuerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

h_2_bi verfügt über eine Liste von Typenumwandlungen, mit der die C-Standard-Variablentypen in die entsprechenden FreeBasic-Variablentypen gewandelt werden, vgl. Tabelle 8.3 auf Seite 34. Diese Liste kann bei Bedarf mit benutzerspezifischen Typumwandlungen ergänzt werden, indem in der Steuerdatei in der Sektion `__TYPES__(){};` entsprechende Declarationen definiert werden.

Wenn die Verwendung der internen Liste abgeschaltet wird (vgl. Kap. 5.6.7 auf Seite 18), dann werden nur die Typumwandlungen der Steuerdatei verwendet, jedoch mit einer Ausnahme: h_2_bi benötigt intern zur Unterscheidung zwischen Prozeduren (SUB) und Funktionen die Definition `void>ANY`, welche sich daher nicht abschalten läßt.

Tabelle 5.1: Beispiel für die Wirkung des Schalters `CTY`

C-Header	`MeinTyp Test(void);`
Steuer-datei	`__TYPES__(){};` `MeinTyp>INTEGER`
`-P_cTY`	`DECLARE FUNCTION Test CDECL() AS MeinTyp`
`-P_CTY`	`DECLARE FUNCTION Test CDECL() AS INTEGER`

5.6.3 .

Die Abkürzung dieses Schalters steht für **.** **TY**pe-Replacements. Mit ihm wird die standard Typenumwandlung geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

Tabelle 5.2: Beispiel für die Wirkung des Schalters `CT`

`-P_cT`	
`-P_CT`	

5.6.4 Gruppe I

Die Kennung `I` steht für „Iutput", mit dieser Option werden die Funktionen zur Generierung der C-Quelltext Kommentare in der Ergebnisdatei gesteuert. Die Funktionen können entweder einzeln durch die Verwendung der Option `-P` geschaltet werden, oder alle Schalter werden gleichzeitig mit der Option `-I` bedient, vgl. Tabelle 5.6.4 auf Seite 17.

Die Schalter haben folgende Bedeutung:

5.6.5 .

Die Abkürzung dieses Schalters steht für **.** **TY**pe-Replacements. Mit ihm wird die standard Typenumwandlung geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.6 auf Seite 37.

h_2_bi verfügt über eine Liste von Typenumwandlungen, mit der die C-Standard-Variablentypen in die entsprechenden FreeBasic-Variablentypen gewandelt werden, vgl. Tabelle 8.3 auf Seite 34. Sie kann bei Bedarf mit benutzerspezifischen Typwandlungen ergänzt werden.

Wenn die Verwendung der der internen Liste abgeschaltet wird (vgl. Kap. **??** auf Seite **??**), dann werden nur die Typumwandlungen der Steuerdatei verwendet, mit einer Ausnahme: h_2_bi benötigt intern zur Unterscheidung zwischen Prozeduren (SUB) und Funktionen die Definition `void>ANY`, welche sich daher nicht abschalten läßt.

Tabelle 5.3: Beispiel für die Wirkung des Schalters `CY`

`-P_cY`	
`-P_CY`	

5.6.6 Gruppe O

Die Kennung `O` steht für „Output", mit dieser Option werden die Funktionen zur Erzeugung des FreeBasic-Quelltextes gesteuert. Die Funktionen können entweder einzeln durch die Verwendung der Option `-P` geschaltet werden, oder alle Schalter werden gleichzeitig mit der Option `-O` bedient, vgl. Tabelle 8.7 auf Seite 38.

Die Schalter haben folgende Bedeutung:

5.6.7 DT

Die Abkürzung dieses Schalters steht für **D**efault **T**ype-Replacements. Mit ihm wird die Standard-Typenumwandlung geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

h_2_bi verfügt über eine Liste von Typenumwandlungen, mit der C-Standard-Variablentypen in die entsprechenden FreeBasic-Variablentypen gewandelt werden, vgl. Tabelle 8.3 auf Seite 34. Die Verwendung der Liste kann bei Bedarf auch abgeschaltet werden, wodurch zunächst keine brauchbare Übersetzung generiert werden kann, da keine Typ-Konvertierung für Variablen mehr durchgeführt wird. Ausgenommen ist die Konvertierung `void>ANY`, welche intern zur Unterscheidung zwischen Prozeduren (SUB) und Funktionen benötigt wird und sich daher nicht abschalten läßt. Um bei agbeschalteter Liste brauchbare Übersetzungen zu erhalten, müssen Typumwandlungen in der Steuerdatei in der Sektion `__TYPES__(){};` definiert werden, vgl. Kap. **??** auf Seite **??**.

Tabelle 5.4: Beispiel für die Wirkung des Schalters `ODT`

C-Header	`int Test;`
`-P_oDT`	`TYPE Test AS int`
`-P_ODT`	`TYPE Test AS INTEGER`

Es wird empfohlen, diesen Schalter generell einzuschalten, da die Definition einer benutzerspezifischen Typumwandlungsliste nicht trivial ist.

5.6.8 DF

Die Abkürzung dieses Schalters steht für **C**onfig **F**ile **FI**le-Replacements. Mit ihm wird die Datei-Ersetzung geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

h_2_bi verfügt über eine Liste von Standard-Headerdateien, welche bereits fertig übersetzt im Lieferumfang von FreeBasic vorhanden sind, vgl. Tabelle 8.2 auf Seite 34. Diese Header müssen gewöhnlich nicht erneut übersetzt werden, sondern können als fertige `.bi`-Datei eingebunden werden.

Die Verwendung dieser Liste ist standardmäßig eingeschaltet. Sie kann bei Bedarf sowohl abgeschaltet als auch ergänzt werden. Die Ergänzung erfolgt durch Einträge in der Steuerdatei unter der Sektion `__BI_REPS__(){};`, vgl. Kap. 4.1.2 auf Seite 12. Die Einträge der Steuedatei werden bevorzugt verwendet. Bei Gleichheit zweier Header-Namen wird also die Datei-Ersetzung aus der Steuerdatei verwendet.

Tabelle 5.5: Beispiel für die Wirkung des Schalters `ODF`

C-Header	`#include <stdio.h>;`
`-P_oDF`	`' 01file not found ???: stdio.h`
`-P_ODF`	`#INCLUDE ONCE "crt/stdio.bi"`

Es wird empfohlen, diesen Schalter generell einzuschalten.

CFST

Die Abkürzung dieses Schalters steht für **C**onfig **F**ile **ST**art-Section. Mit ihm wird die Verwendung der Sektion `__START_BI__(){};` der Steuerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

Standardmäßig liest h_2_bi die Zeilen unter dieser Sektionsüberschrift und überträgt sie unverändert und inklusiv der Leerzeilen an den Anfang der Ergebnisdatei, vgl. Kap. 4.1.2 auf Seite 12.

Tabelle 5.6: Beispiel für die Wirkung des Schalters `CFST`

Steuer-datei	`__START_BI__(){};`
	`' FreeBasic-Befehle oder -kommentare`
	
	`#IFNDEF MeinHeader`
	`#DEFINE MeinHeader`
	`#INCLIB "MeinHeader"`

`-P_cFST`	

`-P_CFST`	`' FreeBasic-Befehle oder -kommentare`
	
	`#IFNDEF MeinHeader`
	`#DEFINE MeinHeader`
	`#INCLIB "MeinHeader"`

CFEN

Die Abkürzung dieses Schalters steht für **C**onfig **F**ile **EN**d-Section. Mit ihm wird die Verwendung der Sektion `__END__(){};` der Steuerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

h_2_bi liest in der Steuerdateidie Zeilen unter der Überschrift der Sektion `__END__(){};` und überträgt sie unverändert und inklusiv aller Leerzeilen an das Ende der Ergebnisdatei– hinter das Übersetzungsergebnis und vor das Protokoll, vgl. Kap. 4.1.2 auf Seite 12. Dies kann bei Bedarf mit diesem Schalter ausgeschaltet werden.

Tabelle 5.7: Beispiel für die Wirkung des Schalters `CFEN`

Steuer-datei	`__END_BI__(){};`
	`' FreeBasic-Befehle oder -kommentare`
	
	`#ENDIF ' MeinHeader`

`-P_cFEN`	

`-P_CFEN`	`' FreeBasic-Befehle oder -kommentare`
	
	`#ENDIF ' MeinHeader`

CFPA

Die Abkürzung dieses Schalters steht für **C**onfig **F**ile **PA**rameter. Mit ihm wird die Verwendung der zusätzlichen Parameter in der Steuerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

h_2_bi verwendet immer diejenigen Parameter, welche in der Kommandozeile angegeben werden. Zusätzlich können weitere Parameter in der Steuerdatei unter der Überschrift der Sektion `__PARAMETERS__(){};` definiert werden, vgl. Kap. 4.1.2 auf Seite 11. Die zusätzlichen Parameter der Steuerdatei werden nach denen der Kommandozeile ausgewertet, heben also ggf. zuvor getätigte Einstellungen wieder auf. Wenn dieser Schalter ausgeschaltet ist, werden keine zusätzlichen Parameter aus der Steuerdateiausgewertet.

Tabelle 5.8: Beispiel für die Wirkung des Schalters `CFPA`

`-P_cFPA`	
`-P_CFPA`	

CFFO

Die Abkürzung dieses Schalters steht für **C**onfig **F**ile **FO**lder. Mit ihm wird die Verwendung der Zugriffspfade in der Sektion `__PATHS__(){};` der Steuerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

h_2_bi sucht nach C-Headerdateien zunächst im aktuellen Arbeitsverzeichnis. Wird eine Datei dort nicht gefunden, so werden diejenigen Zugriffspfade durchsucht, welche in der Steuerdatei in der Sektion `__PATHS__(){};` definiert sind, vgl. Kap. 4.1.2 auf Seite 12. Die Suche erfolgt in der Reihenfolge der Pfadangaben. h_2_bi verwendet die zuerst gefundene Datei.

Wenn diese Funktion ausgeschaltet ist, sucht h_2_bi C-Headerdateien nur im aktuellen Arbeitsverzeichnis. Falls eine Headerdatei dort nicht existiert, schreibt h_2_bi eine Meldung in die Ergebnisdatei. Andernfalls, also wenn der korrekte Zugriffspfad in der Steuerdatei definiert ist und diese Funktion eingeschaltet ist, wird die Headerdatei gelesen und deren Übersetzung in die Ergebnisdatei übernommen.

In der Protokollsektion der Ergebnisdatei werden alle Dateinamen mit Pfadangabe aufgelistet, welche nicht gefunden werden konnten. Anhand dieser Liste können die notwendigen Einträge in die Steuerdatei vorgenommen werden. Einerseits können Pfad-Einträge absolut erfolgen, also als vollständiger Pfad beginnend mit dem Laufwerksbezeichner (z. B. `/home/ich/Projekte/GTK/src` oder `Z:\Projekte\GTK\src`). Andererseits kann der Pfad auch relativ zum aktuellen Arbeitsverzeichnis angegeben werden (siehe Beispiel), was die Portierung der Daten auf andere EDV-Systeme vereinfacht.

Es ist zu beachten, dass eingebundene C-Headerdateien ebenfalls weitere Header einbinden können. Nachdem also die Liste der Zugriffspfade in der Sektion `__PATHS__(){};` gemäß den Einträgen fehlender Dateien in der Protokoll-Sektion der Ergebnisdatei aufgefüllt wurde, können beim nächsten Lauf von h_2_bi durchaus neue fehlende Dateien im Protokoll aufgelistet werden, weil mit der Bearbeitung neu eingebundener Header weitere Zugriffspdfade notwendig werden.

Tabelle 5.9: Beispiel für die Wirkung des Schalters `CFFO`

C-Header	`#include "gdk.bi"`
Steuer-datei	`__PATHS__(){};` `gtk+-2.90.1/gdk/ // Beispielpfad`
`-P_cFFO`	`' 01file not found ???: gdk.h`
`-P_CFFO`	`' 01file: gtk+-2.90.1/gdk/gdk.h ' + Uebersetzung`

CFMA

Die Abkürzung dieses Schalters steht für **C**onfig **F**ile **MA**cros. Mit ihm wird die Verwendung der Makro-Definitionen in der Sektion `__MACROS__(){};` der Steuerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

h_2_bi expandiert Makros im C-Quelltext, wenn diese in der Steuerdatei definiert sind, vgl. Kap. 4.1.2 auf Seite 13. Alle zur Übersetzung der C-Headerdatei definierten Makros werden in der Protokollsektion der Ergebnisdatei aufgelistet. Nicht benötigte Makros können anhand einer Einrückung bzw. an dem Zählerstand `0` erkannt werden, vKRFNO.

Die Expansion der Makros kann unterdrückt werden, indem dieser Schalter abgeschaltet wird.

Tabelle 5.10: Beispiel für die Wirkung des Schalters `CFMA`

C-Header	`MY_AR(char, Test);`
Steuer-datei	`__MACROS__(){};` `#define MY_AR(type, name) extern type name[]`
`-P_cFMA`	`DECLARE FUNCTION char, Test CDECL() AS MY_AR`
`-P_CFMA`	`TYPE Test AS ZSTRING PTR`

CFFB

Die Abkürzung dieses Schalters steht für **C**onfig **F**ile **F**ee**B**asic-Quelltext. Mit ihm wird die Verwendung der FreeBasic-Ersetzungen in der Sektion `__BASIC_REPS__(){};` der Steuerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.5 auf Seite 36.

h_2_bi kann einzelne Worte im FreeBasic-Quelltext der Ergebnisdatei ersetzen oder ergänzen. Dazu werden Textzeilen mit dem Suchwort und dessen Ersetzung in der Sektion `__BASIC_REPS__(){};` in der Steuerdatei definiert, vgl. Kap. 4.1.2 auf Seite 13. Die Ausführung der Ersetzungsfunktion kann mit diesem Schalter unterbunden werden.

Tabelle 5.11: Beispiel für die Wirkung des Schalters `CFFB`

C-Header	`typedef struct _GTree GTree;`

Steuer-datei	`__BASIC_REPS__(){};`
	`TYPE>Type // Schreibweise TYPE-Befehl`
	`AS>As // Schreibweise AS-Befehl`

`-P_cFFB`	`TYPE GTree AS _GTree`
`-P_CFFB`	`Type GTree As _GTree`

RFPR

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **PR**otocol. Mit ihm wird die Funktion zur Erstellung eines Protokolles in der Ergebnisdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

h_2_bi erzeugt am Ende der Ergebnisdatei eine Sektion, in welcher ein Protokoll von dem Übersetzungsvorgang ausgegeben wird, vgl. Kap. 8.2 auf Seite 33. Die Ausgabe des Protokolles kann mit diesem Schalter unterbunden werden.

Tabelle 5.12: Beispiel für die Wirkung des Schalters `RFPR`

`-P_rFPR`	

`-P_RFPR`	`' Protocol: src.bi`
	`' Parameters:`
	`...`

RFCO

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **CO**mpress. Mit ihm wird die Funktion zur Komprimierung des FreeBasic-Quelltext geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

...

Tabelle 5.13: Beispiel für die Wirkung des Schalters `RFCO`

`-P_rFCO`	
`-P_RFCO`	

RFNO

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **NO**mber of use. Mit ihm wird die Funktion zur Anzeige der Zähler im Protokoll der Ergebnisdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

h_2_bi erzeugt am Ende der Ergebnisdatei eine Protokollsektion, falls der Schalter `RFPR` eingeschaltet ist, vgl. Kap. 5.6.8 auf Seite 21 und vgl. Kap. 8.2 auf Seite 33. Innerhalb des Protokolles werden die Einträge der Steuerdatei aufgelistet, welche in den Sektionen `MACROS`, `BASIC_REPS` und `TYPES` enthalten sind. h_2_bi zählt, wie häufig jeder dieser Einträge beim Übersetzungsvorgang verwendet wird und gibt die Zählerstände am Anfang der jeweiligen Zeile aus. Die Zahlenausgabe kann mit diesem Schalter unterbunden werden.

Auch bei abgeschalteter Zahlenausgabe führt h_2_bi eine Kennzeichnung durch, indem Einträge, welche nicht benutzt wurden, durch ein Leerzeichen eingerückt werden.

Tabelle 5.14: Beispiel für die Wirkung des Schalters RFNO

-P_rFNO	
-P_RFNO	

RFWR

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **WR**app parameter lists. Mit ihm wird die Funktion zum Zeilenumbruch in Parameterlisten geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

h_2_bi übersetzt Parameterlisten, welche bei der Definition von Funktionen in der C-Headerdatei vorkommen. Solche Parameterlisten können gegebenenfalls sehr lang und damit unübersichtlich werden. Um eine bessere Lesbarkeit des FreeBasic-Quelltextes zu erreichen, ist eine Zeilenumbruchfunktion vorhanden, welche jeden Parameter der Liste jeweils in eine eigene Zeile setzt. Die einzelnen Zeilen werden für den FreeBasic-Compilermit dem Zeilenverbindungszeichen abgeschlossen (_).

Falls dieser Schalter abgeschaltet ist, wird die gesamte Parameterliste in einer Zeile ausgegeben.

Tabelle 5.15: Beispiel für die Wirkung des Schalters RFWR

C-Header	`typedef void (*GPoll) (GPollFD *ufds);`
-P_rFWR	`TYPE GPoll AS SUB CDECL(BYVAL ufds AS GPollFD PTR)`
-P_RFWR	`TYPE GPoll AS SUB CDECL( BYVAL ufds AS GPollFD PTR) AS gint`

RFNA

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **NA**mes in parameter lists. Mit ihm wird die Funktion zur Übernahme der Variablen-Namen in Parameterlisten geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

h_2_bi übersetzt Parameterlisten, welche bei der Definition von Funktionen in der C-Headerdatei vorkommen. Solche Parameterlisten können Namensbezeichner enthalten, welche Aufschluß über die Verwendung der Variablen geben (z. B. Name in (`BYVAL Name AS STRING`)). Diese Namensbezeichenr sind in einer Headerdatei weder für die Programmiersprache FreeBasic noch für C notwendig, weshalb sie in C-Headerdateien teilweise auch weggelassen werden.

Für die Interpretation der in der Parameterlist übergebenen Variablen kann die Verwendung des Namensbezeichners hilfreich sein. h_2_bi kann Namensbezeichner nur dann in Parameterlisten einfügen, wenn diese auch in der C-Headerdatei definiert sind.

Falls dieser Schalter abgeschaltet ist, wirden keine Namensbezeichenr in die Parameterlisten übernommen. Die erzeugte FreeBasic-Headerdatei benötigt weniger Speicherplatz und deren Übersetzung durch den FreeBasic-Compiler erfolgt entsprechend schneller.

Tabelle 5.16: Beispiel für die Wirkung des Schalters RFNA

C-Header	`typedef void (*GPoll) (GPollFD *ufds);`
-P_rFNA	`TYPE GPoll AS SUB CDECL(BYVAL ufds AS GPollFD PTR)`
-P_RFNA	`TYPE GPoll AS SUB CDECL(BYVAL AS GPollFD PTR)`

RFBY

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **BY**val in parameter lists. Mit ihm wird die Funktion zur Erzeugung von **BYVAL**-Befehlen in Parameterlisten geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

h_2_bi übersetzt Parameterlisten, welche bei der Definition von Funktionen in der C-Headerdatei vorkommen. In solchen Parameterlisten werden die Variablen gewöhnlich als Kopie auf dem Stack übergeben, was mit dem FreeBasic-Befehl **BYVAL** innerhalb der Parameterliste ausgedrückt wird. In neueren FreeBasic-Dialekten kann auf die Verwendung des **BYVAL**-Befehles verzichtet werden, indem die Übergabe der Parameter mit einer Zeile `#LANG "fb"` generell auf den Typ **BYVAL** eingestellt wird. Dadurch werden die Parameterlisten kürzer und die FreeBasic-Headerdatei benötigt weniger Speicherplatz.

Die Verwendung der Option `#LANG "fb"` kann mit diesem Schalter abgeschaltet werden, z. B. aus Kompatibilitätsgründen, wenn mit FreeBasic-Compilern vor Versionsnummer 0.2 gearbeitet werden soll.

Tabelle 5.17: Beispiel für die Wirkung des Schalters `RFBY`

C-Header	`typedef void (*GPoll) (GPollFD *ufds);`
`-P_rFBY`	`TYPE GPoll AS SUB CDECL(BYVAL ufds AS GPollFD PTR)`

`-P_RFBY`	`#LANG "fb"`
	`...`
	`TYPE GPoll AS SUB CDECL(ufds AS GPollFD PTR)`

RFCD

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **CD**ecl befor parameter lists. Mit ihm wird die Funktion zur Erzeugung von **CDECL**-Befehlen vor Parameterlisten geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

h_2_bi übersetzt Parameterlisten, welche bei der Definition von Funktionen in der C-Headerdatei vorkommen. Für die korrekte Übergabe der Parameter vom FreeBasic-Quelltext an die C-Bibliothek sorgt der **CDECL**-Befehl, welcher vor der Parameterliste angegeben wird.

In neueren FreeBasic-Dialekten kann auf die Verwendung des **CDECL**-Befehles auch verzichtet werden, indem die Funktionsdefinitionen, für welche die Parameterübergabe mit dem Typ **CDECL** erfolgen soll, in einen Block eingeschlossen werden, welcher mit der Zeile `EXTERN "C"` beginnt und mit der Zeile **END EXTERN** abgeschlossen wird. Die Parameterübergaben für Funktionen und Prozeduren (**SUB**) innerhalb des Blockes wird dann automatisch mit dem **CDECL**-Befehl übersetzt, was im Falle von h_2_bi für alle C-Headerdateien, also die gesamte Übersetzung der Ergebnisdatei gilt. Dadurch werden die Parameterlisten kürzer und die FreeBasic-Headerdatei benötigt weniger Speicherplatz.

Die Verwendung der `EXTERN "C"`-Blöcke ist nur im FreeBasic-Dialekt erlaubt. Daher verwendet h_2_bi vor dem Block eine weitere Zeile, welche diese Dialektform einschaltet (oder bei unzureichender FreeBasic-Compiler-Version eine Fehlermeldung ausgibt).

Die Verwendung des Blockes `EXTERN "C"` kann mit diesem Schalter abgeschaltet werden, z. B. aus Kompatibilitätsgründen, wenn mit FreeBasic-Compiler n vor Versionsnummer 0.2 gearbeitet werden soll.

Tabelle 5.18: Beispiel für die Wirkung des Schalters `RFCD`

C-Header	`typedef void (*GPoll) (GPollFD *ufds);`
`-P_rFCD`	`TYPE GPoll AS SUB CDECL(BYVAL ufds AS GPollFD PTR)`

`-P_RFCD`	`#LANG "fb"`
	`EXTERN "C"`
	`...`
	`TYPE GPoll AS SUB (BYVAL ufds AS GPollFD PTR)`
	`...`
	`END EXTERN`

RFNT

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **N**ull **T**o. Mit ihm wird die Funktion zur Dimensionierung von Feldern mit der Zeichenfolge `0 TO` geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Die Dimensionierung von Feldern (Arrays) erfolgt in den Programmiersprachen C und FreeBasic unterschiedlich. In C wird die Feldgröße in geschweiften Klammern angegeben, z. B. `Name[9]` für ein Feld mit 9 Einträgen, welche mit dem Index 0 bis 8 angesprochen werden. Ein entsprechendes Feld wird im FreeBasic-Quelltext mit **`Name`**`(8)` dimensioniert. Die unterschiedlichen Zahlen in der Dimensionierung sind bei der Überprüfung des Übersetzungsergebnisses irreführend. Aus Bequemlichkeitsgründen bietet h_2_bi deshalb die Möglichkeit, eine alternative Dimensionierung im FreeBasic-Quelltext zu verwenden, bei der die Dimensionierung aus dem C-Header leichter erkennbar ist, vgl. Tabelle 5.6.8 auf Seite 24.

Mit diesem Schalter kann die Verwendung der Langform **`Name`**`(0 TO xx-1)` abgeschaltet werden, wodurch die weniger übersichtliche Kurzform **`Name`**`(xx)` zum Einsatz kommt.

Tabelle 5.19: Beispiel für die Wirkung des Schalters `RFNT`

C-Header	**struct** _MyMonth
	{
	GtkMisc Monate[12];
	};

-P_rFNT	**TYPE** _MyMonth **AS** GtkMisc Monate(11) **END TYPE**
-P_RFNT	**TYPE** _MyMonth **AS** GtkMisc Monate(0 TO 12-1) **END TYPE**

RFTE

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **T**ype **E**xtension. Mit ihm wird die Funktion zur Ergänzung **`TYPE`**-Befehlen mit identischem Namen und Typ geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

In C-Headerdateien werden sogenannte Vorwärts-Deklarationen verwendet, mit welchen einem Variablennamen ein Varaiblentyp zugewiesen wird. In der Programmiersprache C ist es dabei zulässig, das der Name und der Typ identisch bezeichnet sind, was bei direkter Übersetzung in den entsprechenden FreeBasic-Quelltext zu einer Fehlermeldung des FreeBasic-Compiler führt. Um solche Namenskonflikte zu vermeiden, kann h_2_bi den Typ der Deklaration mit der Zeichenfolge `_TJF` ergänzen. Diese Ergänzung erfolgt sowohl in der Vorwärts-Deklaration als auch bei der späteren Definition des Typen.

Mit diesem Schalter kann die Ergänzung von Typnamen in Vorwärts-Deklarationen und Typdefinitionen abgeschaltet werden.

Tabelle 5.20: Beispiel für die Wirkung des Schalters `RFTE`

C-Header	;
-P_rFTE	x
-P_RFTE	x

RFTN

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **T**ype **N**aming. Mit ihm wird die Funktion zur automatischen Benennung von unbenannten Variablen geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

...

Tabelle 5.21: Beispiel für die Wirkung des Schalters `RFTN`

C-Header	`;`
Steuer-datei	`__x__(){};` `x`
`-P_rFTN`	`x`
`-P_RFTN`	`x`

Tabelle 5.22: Beispiel für die Wirkung des Schalters `RFSF`

C-Header	`typedef void (*a)(void);` `typedef void (*b)(c *a);`
`-P_rFSF`	`TYPE a AS SUB CDECL()` `TYPE b AS SUB CDECL(BYVAL c AS a PTR)`
`-P_RFSF`	`TYPE a AS SUB CDECL()` `TYPE b AS SUB CDECL(BYVAL c AS a)`

RFSF

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **S**ubs and **Functions**. Mit ihm wird die Funktion zur Anpassung von Funktionsaufrufen durch einen Pointer geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Die Syntax des Aufrufes von Prozeduren (**SUB**) und Funktionen in den Programmiersprachen C und FreeBasic ist unterschiedlich. Während in der C-Syntax durch einen `*` kenntlich gemacht wird, dass die Funktion über einen Zeiger aufgerufen wird, fehlt in der FreeBasic-Syntax der entsprechende **PTR**-Befehl. h_2_bi kann diese Syntaxunterschiede übersetzen, indem bei allen Typedefinitionen in **TYPE**-Zeilen oder -Blöcken, sowie in Parameterlisten ein **PTR**-Befehl der regulären Übersetzung entfernt wird, sofern es sich bei dem Typnamen um eine zuvor definierte **SUB** oder **FUNCTION** handelt.

Wenn dieser Schalter abgeschaltet ist, erfolgt eine reguläre Übersetzung, bei welcher der Funktionsname wie ein gewöhnlicher Variablenname behandelt wird.

RFRM

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **R**ecursiv **M**acro expansion. Mit ihm wird die Funktion zur Expansion von Makros innerhalb von Makrotexten geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Makro-Definitionen bestehen aus einem Makro-Namen, ggf. einer Parameterliste und ggf. auch aus einem Makro-Text. In einem Makro-Text kann der Aufruf eines (oder mehrerer) weiteren (er) Makros enthalten sein, was Verschachtelung genannt wird. h_2_bi kann sowohl ein einfaches Makro als auch ineinander verschachtelte Makros expandieren, sofern diese in der Sektion `__MACROS__(){};` in der Steuerdatei definiert sind, vgl. Kap. 4.1.2 auf Seite 13.

Verschachtelte Makro werden jedoch nicht beliebig expandiert. Wenn innerhalb einer Makroverschachtelung ein zuvor bereits verwendetes Makro erneut aufgerufen würde, dann würde eine Endlosschleife entstehen (h_2_bi „hängt sich auf"). Um dies zu vermeiden, wird die Expansion von Makros nur solange vorgenommen. solange der Name von inneren Makro in der zuvor durchgeführten Expansion noch nicht vorgekommen ist.

Wenn dieser Schalter abgeschaltet ist, werden nur einfache Makros expandiert und keine Expansionen verschachtelter Makros vorgenommen.

RFCA

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **CA**st translations. Mit ihm wird die Funktion zur Übersetzung von **CAST**-Befehlen geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Tabelle 5.23: Beispiel für die Wirkung des Schalters `RFRM`

C-Header	`;`
Steuer-datei	`__x__(){};` `x`
`-P_rFRM`	`x`
`-P_RFRM`	`x`

Die Syntax von **`CAST`**-Befehlen in den Programmiersprachen C und FreeBasic ist unterschiedlich. h_2_bi übersetzt diese in einigen Präkompilerzeilen (`#`**`IF`**, `#`**`ELSEIF`**, `#`**`DEFINE`**) und in den Zeilen von **`ENUM`**-Blöcken, vgl. Tabelle 8.1 auf Seite 34.

Wenn dieser Schalter abgeschaltet ist, werden keine Übersetzungen der **`CAST`**-Syntax vorgenommen.

Tabelle 5.24: Beispiel für die Wirkung des Schalters `RFCA`

C-Header	**`#define`** `ty_co(a, b) ((a) b)`
`-P_rFCA`	`#`**`DEFINE`** `ty_co(a, b) ((a) b)`
`-P_RFCA`	`#`**`DEFINE`** `ty_co(a, b) (`**`CAST`**`(a, b))`

RFII

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **II**f translations. Mit ihm wird die Funktion zur Übersetzung von **`IIF`**-Befehlen geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Die Syntax von **`IIF`**-Befehlen in den Programmiersprachen C und FreeBasic ist unterschiedlich. h_2_bi übersetzt diese in einigen Präkompilerzeilen (`#`**`IF`**, `#`**`ELSEIF`**, `#`**`DEFINE`**) und in den Zeilen von **`ENUM`**-Blöcken, vgl. Tabelle 8.1 auf Seite 34.

Wenn dieser Schalter abgeschaltet ist, werden keine Übersetzungen der **`IIF`**-Syntax vorgenommen.

Tabelle 5.25: Beispiel für die Wirkung des Schalters `RFII`

C-Header	**`#define`** `MyMAX(x, y) x > y ? x : y`
`-P_rFII`	`#`**`DEFINE`** `MyMAX(x, y) x > y ? x : y`
`-P_RFII`	`#`**`DEFINE`** `MyMAX(x, y)` **`IIF`**`(x > y, x, y)`

RFQU

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **QU**ote translations. Mit ihm wird die Funktion zur Übersetzung der Anführungszeichen geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Die Syntax von Anführungszeichen in den Programmiersprachen C und FreeBasic ist unterschiedlich. h_2_bi übersetzt diese in einigen Präkompilerzeilen (`#`**`IF`**, `#`**`ELSEIF`**, `#`**`DEFINE`**) und in den Zeilen von **`ENUM`**-Blöcken, vgl. Tabelle 8.1 auf Seite 34. Es werden sowohl einfache Anführungszeichen (`'`) als auch doppelte Anführungszeichen (`"`) berücksichtigt.

Wenn dieser Schalter abgeschaltet ist, werden keine Übersetzungen der Anführungszeichen-Syntax vorgenommen.

RFLO

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **LO**gic translations. Mit ihm wird die Funktion zur Übersetzung der Logik-Befehle in einer C-Headerdatei geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Tabelle 5.26: Beispiel für die Wirkung des Schalters RFQU

C-Header	`#define K_bracketright ']'` `#define a_string "Some\nText!"`
-P_rFQU	`#DEFINE K_bracketright 'ʃ'` `#DEFINE a_string "Some\nText!"`
-P_RFQU	`#DEFINE K_bracketright Asc(!"]")` `#DEFINE a_string !"Some\nText!"`

Die Syntax von Logik-Befehlen in den Programmiersprachen C und FreeBasic ist unterschiedlich. h_2_bi übersetzt diese in einigen Präkompilerzeilen (**#IF**, **#ELSEIF**, **#DEFINE**) und in den Zeilen von **ENUM**-Blöcken, vgl. Tabelle 8.1 auf Seite 34.

Wenn dieser Schalter abgeschaltet ist, werden keine Übersetzungen der Logik-Syntax vorgenommen.

Tabelle 5.27: Beispiel für die Wirkung des Schalters RFLO

C-Header	`#if defined (G_OS_WIN) && GLIB_SIZE == 072`
-P_rFLO	`#IF DEFINED (G_OS_WIN) && GLIB_SIZE == 072`
-P_RFLO	`#IF DEFINED (G_OS_WIN) AND GLIB_SIZE = &o72`

RFAN

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **AN**y typedefs. Mit ihm wird die Funktion zur Verwendung des ANY-Typen geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Die C-Syntax erlaubt es, Variablen ohne Typzuweisung zu definieren. In FreeBasic ist es dagegen vorgesehen, in jeder Variablendefinition auch eine Typzuweisung vorzusehen, wobei auch ein beliebiger Typ namens ANY verwendet werden kann. h_2_bi verwendet diesen beliebigen Typen standardmäßig.

Wenn dieser Schalter jedoch abgeschaltet ist, wird einer entsprechenden FreeBasic-Variablen wie im C-Header kein Typ zugewiesen, was bei der Kompilierung der Ergebnisdatei mit dem FreeBasic-Compilerdann zu einer Fehlermeldung führt.

Tabelle 5.28: Beispiel für die Wirkung des Schalters RFAN

C-Header	`typedef EinName;`

-P_rFAN	`TYPE EinName AS`
	
	`' Anm.:`
	`' Diese Zeile erzeugt beim Kompilieren`
	`' mit dem fbc einen Fehler`

-P_RFAN	`TYPE EinName AS ANY`

RFUC

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **U**nsave **C**asts. Mit ihm wird die Übersetzung von unsicheren **CAST**-Befehlen geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Im C-Quelltext wird eine Typenwandlung (**CAST**) durch einen eingeklammerten Variablentypen vor dem Wert formuliert. Der C-Kompiler kann diese Syntax sicher interpretieren, da er alle verfügbaren Variablentypen kennt. In h_2_bi wird dagegen bewußt auf eine Typenprüfung verzichtet („dumme" Übersetzung), wodurch die sichere Erkennung von Typenumwandlungen im C-Quelltext schwierig wird.

Bei der Entwicklung von h_2_bi hat es sich als sinnvoll erwiesen, auch unsichere Typenumwandlungen zu übersetzen, diese jedoch als solche zu Kennzeichnen. Als

Tabelle 5.29: Beispiel für die Wirkung des Schalters `RFUC`

C-Header	`#define MyMACRO(a, b) ((a)*(b));`
`-P_rFUC`	`#DEFINE MyMACRO(a, b) (CAST(a, b PTR))`
`-P_RFUC`	`#DEFINE MyMACRO(a, b) (CAST(a, b PTR)) ' ??? unsave CAST` `' Anm.: Bei diesem Beispiel wird davon ausgegangen, dass der Schalter RFET eingeschaltet ist.`

unsichere **CAST**s werden solche betrachtet, deren Argument mit einem unären Operator beginnt (`*, &, -, ! oder ~`). Der Anwender kann dann die Sinnhaltigkeit der Übersetzung prüfen und ggf. den **CAST**-Befehl wieder aus der Ergebnisdatei entfernen. In einfachen Fällen wird dazu das Wort **CAST** gelöscht, das folgende Komma in eine schließende Klammer (')') gewandelt und die folgende schließende Klammer gelöscht. Schwieriger ist die Entfernung eines **CAST**s, wenn das Argument mit einem der unären Operatoren `*` oder `&` beginnt.

Wenn dieser Schalter ausgeschaltet wird, dann werden unsichere **CAST**-Befehle nicht gekennzeichnet.

RFMB

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **M**ark removed **B**locks. Mit ihm wird die Kennzeichnung von unübersetzten C-Quelltextblöcken geschaltet. Standardmäßig ist er abgeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Wenn h_2_bi beim Übersetzungsvorgang auf Quelltextblöcke trifft, welche für die FreeBasic-Übersetzung keine Bedeutung haben, dann werden diese Blöcke nicht übersetzt. Die Ergebnisdatei bleibt an dieser Stelle ganz leer, sofern nicht der Quelltext übernommen wird, weil die Generierung von C-Kommentar eingeschaltet ist. Es kann, z. B. zur Orientierung in der Ergebnisdatei sinnvoll sein, dass die Position eines entfernten Blockes gekennzeichnet wird. Wenn dieser Schalter eingeschaltet wird, schreibt h_2_bi an der Position eines entfernten Blockes einen Kommentar.

Derzeit entfernt h_2_bi C-Quelltextblöcke, die mit **static** oder `inline` beginnen.

Tabelle 5.30: Beispiel für die Wirkung des Schalters `RFMB`

C-Header	`inline *char`
	`func(void){`
	`return "nix";`
	`};`

`-P_rFMB`	
`-P_RFMB`	`' removed inline block`

RFMI

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **M**ark lines in **I**f-Blocks. Mit ihm wird die Kennzeichnung von Präkompiler **#if**-Blöcken geschaltet. Standardmäßig ist er abgeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Die Kompilierung einer oder mehrerer Header-Dateien kann durch die Verwendung von **#if**-Blöcken gesteuert werden, wobei einzelne Quelltextzeilen nur unter bestimmten Bedingungen kompiliert werden. Es gibt große Unterschiede zwischen den für C und FreeBasic relevanten Bedingungen. Oftmals kann ein Großteil der im C-Header vorhandenen Quelltextzeilen im FreeBasic-Header gelöscht werden.

Die **#if**-Blöcke, welche zur bedingten Kompilierung eingesetzt werden, erstrecken sich nicht selten über viele huntert Zeilen und sind auch oft ineinander verschachtelt, so dass es schwierig ist, die Struktur zu überblicken. Das Programm h_2_bi kann die Orientierung im FreeBasic-Quelltext erleichtern, indem es Markierungen als Kommentar in die Ergebnisdatei eingefügt werden.

Wenn dieser Schalter eingeschaltet ist, werden Markierungen bei Blöcken angebracht, die mit den Befehlen **#ifdef** oder **#ifndef** beginnen. Als Markierungskommentar wird die Bedingung des Befehles an den zugehörigen **#endif**-Befehl und - falls vorhanden - auch an einen zugehörigen **#else**-Befehl angehängt.

Tabelle 5.31: Beispiel für die Wirkung des Schalters `RFMI`

C-Header	**#ifndef** MeinHeader ... **#else** ... **#endif**
	#**IFNDEF** MeinHeader ... #**ELSE** ... #**ENDIF**

RFEB

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **E**rror codes **B**inär. Mit ihm wird die Meldung von Fehlern als Binärzahl geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Wenn h_2_bi beim Übersetzungsvorgang auf unerwarteten Inhalt in einer C-Headerdatei trifft, dann wird dennoch versucht, einen sinnvollen FreeBasic-Quelltext daraus zu generieren. Diese zweifelhafte Quelltextzeile wird mit drei Fragezeichen `???` markiert, welche durch eine Fehlermeldung ergänzt werden können. Wenn dieser Schalter eingeschaltet ist, wird die Fehlermeldung in einer Kurzform, verschlüsselt als Binärzahl ausgegeben, vgl. Tabelle 8.4 auf Seite 35. Die Binärzahl kann mehrere Fehlermeldungen gleichzeitig beinhalten.

Tabelle 5.32: Beispiel für die Wirkung des Schalters `RFEB`

C-Header	`;`
Steuer-datei	`__x__(){};` `x`
`-P_rFEB`	`x`
`-P_RFEB`	`x`

RFET

Die Abkürzung dieses Schalters steht für **R**esult **F**ile **E**rrors in **T**ext form. Mit ihm wird die Meldung von Fehlern in Textform geschaltet. Standardmäßig ist er eingeschaltet, vgl. Tabelle 8.7 auf Seite 38.

Wenn h_2_bi beim Übersetzungsvorgang auf unerwarteten Inhalt in einer C-Headerdatei trifft, dann wird dennoch versucht, einen sinnvollen FreeBasic-Quelltext daraus zu generieren. Diese zweifelhafte Quelltextzeile wird mit drei Fragezeichen `???` markiert, welche durch eine Fehlermeldung ergänzt werden können.

Wenn dieser Schalter eingeschaltet und der Schalter `CFEB` gleichzeitig ausgeschaltet ist, dann werden die Fehlermeldungen als lesbarer Klartext ausgegeben, vgl. Tabelle 8.4 auf Seite 35.

Tabelle 5.33: Beispiel für die Wirkung des Schalters `RFET`

C-Header	`;`
Steuer-datei	`__x__(){};` `x`
`-P_rFET`	`x`
`-P_RFET`	`x`

5.6.9 Weitere Beeinflussungsmöglichkeiten

5.7 C-Kommentargenerierung (Ergebnisdatei)

5.7.1 Option -C

5.7.2

5.7.3

5.7.4

5.7.5 Weitere Beeinflussungsmöglichkeiten

5.8 Protokoll

6 Anwendungsbeispiel

6.1 Vorbereitungen

6.2 Grundeinstellungen

6.3 Optimierungen

7 Wie geht das?

In diesem Kapitel wird die Anwendung von h_2_bi anhand von verschiedenen Aufgabenstellungen beschrieben.

7.1

7.2

7.3

7.4

7.5

7.6

7.7

7.8

8 Textausgaben/Tabellen

In diesem Kapitel werden Beispiele für Textausgaben von h_2_bi in den Dateien und in der Konsole beschrieben, sowie die Funktionen des Programmes in Tabellenform dargestellt und detailiert erläutert.

8.1 Konsolenausgabe

8.2 Protokoll der Ergebnisdatei

8.3 Programmfunktionen

Tabelle 8.1: Syntaxwandlung für Präcompilerzeilen und Zeilen in **ENUM**-Blöcken.

C-Syntax	FreeBasic-Syntax	Ausnahmen bei #**IF**, #**ELIF**
>>	**SHR**	
<<	**SHL**	
||	**ORELSE**	**OR**
&&	**ANDALSO**	**AND**
==	=	
|=	**OR**=	
&=	**AND**=	
!=	<>	
!	0 =	**NOT**
^	**XOR**	
~	**NOT**	
|	**OR**	
%	**MOD**	
&	**AND**	
0xZIFFERN	&hZIFFERN	
0ZIFFERN	&oZIFFERN	
\ (Zeilenende)	_ (Zeilenende)	

Tabelle 8.2: Standard-Datei Ersetzungen

C-Datei	FreeBasic-Ersatz
wchar.h	crt/wchar.bi
unistd.h	crt/unistd.bi
time.h	crt/time.bi
string.h	crt/string.bi
stdlib.h	crt/stdlib.bi
stdio.h	crt/stdio.bi
stdint.h	crt/stdint.bi
stddef.h	crt/stddef.bi
stdarg.h	crt/stdarg.bi
setjmp.h	crt/setjmp.bi
process.h	crt/process.bi
netdb.h	crt/netdb.bi
math.h	crt/math.bi
malloc.h	crt/malloc.bi
limits.h	crt/limits.bi
io.h	crt/io.bi
fcntl.h	crt/fcntl.bi
errno.h	crt/errno.bi
dir.h	crt/dir.bi
ctype.h	crt/ctype.bi

Tabelle 8.3: Standard-Typumwandlung

C-Typ	FreeBasic-Typ
const	CONST
signed int	INTEGER
unsigned int	UINTEGER
int	INTEGER
signed char	BYTE
unsigned char	UBYTE
char	ZSTRING
signed short	SHORT
unsigned short	USHORT
short	SHORT
unsigned long long	ULONGINT
signed long long	LONGINT
unsigned long	UINTEGER
signed long	INTEGER
long float	DOUBLE
float	SINGLE
long double	DOUBLE
double	DOUBLE
long long	LONGINT
long	INTEGER
unsigned	UINTEGER

Tabelle 8.4: Binärcode für Fehler

Bit-Nr.	Binärwert	Fehlertext	Bedeutung
0	1	`no type`	Kein Variablentyp in Definition
1	10	`no name`	Kein Variablenname in Definition
2	100	`no dimension`	Kein Dimensionierung bei Felddefinition
3	1000	`no parameter list`	Keine Parameterliste bei SUB/FUNCTION-Definition
4	10000	`no type in parameter list`	Kein Variablentyp in Parameterliste
5	100000	`no name in parameter list`	
6	1000000	`no type for SUB/FUNC`	
7	10000000	`no name for SUB/FUNC`	
8	100000000	`undefined character in SUB/FUNC name`	
9	1000000000	`undefined character in line`	
10	10000000000	`undefined block structure in BlockTyUn`	
11	100000000000	`unsave CAST`	

Tabelle 8.5: Funktionen der Option `-c` (1 = Funktion eingeschaltet, 0 = Funktion ausgeschaltet, - = keine Veränderung)

			dezimal		-Level										Optionen					
Nr	**Kürzel**	**Beschreibung**	**Wert**	**Default**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**-f**	**-w**	**-n**			

Tabelle 8.6: Funktionen der Option `-I` (1 = Funktion eingeschaltet, 0 = Funktion ausgeschaltet, - = keine Veränderung)

			dezimal		-Level										Optionen					
Nr	**Kürzel**	**Beschreibung**	**Wert**	**Default**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**-f**	**-w**	**-n**			

Tabelle 8.7: Funktionen der Option `-O` (1 = Funktion eingeschaltet, 0 = Funktion ausgeschaltet, - = keine Veränderung)

			dezimal		-Level										Optionen					
Nr	**Kürzel**	**Beschreibung**	**Wert**	**Default**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**-f**	**-w**	**-n**			
0	PR	erzeuge Protokoll in der Ergebnisdatei	1	1											-	-	-	-	-	-
1	NO	gebe Zählerstände im Protokoll aus	2	1											-	-	-	-	-	-
2	CO	Komprimierung von **`#define`**-Zeilen	4	1											-	-	-	-	-	-
3	WR	Zeilenumbruch in Parameterlisten	8	1											-	-	-	-	-	-
4	NA	Variablen-Namen in Parameterlisten	16	1											-	-	-	-	-	-
5	BV	verwende **`BYVAL`** in Parameterlisten	32	1											-	-	-	-	-	-
6	CD	verwende **`CDECL`** vor Parameterlisten	64	1											-	-	-	-	-	-
7	AL	verwende `ALIAS` für `SUB`/`FUNCTION`	128	1											-	-	-	-	-	-
8	NT	verwende `0 TO` für Feld-Dimensionierung	256	1											-	-	-	-	-	-
9	UT	verwende `_TJF` für `type a a;`	512	1											-	-	-	-	-	-
10	UV	verwende `_TJF` für unbenannte Variablen	1024	1											-	-	-	-	-	-
11	KW	verwende von `_TJF` for FB-Schlüsselwörter	2048	1											-	-	-	-	-	-
12	PT	entferne **`PTR`** bei **`SUB`**/**`FUNCTION`**	4096	1											-	-	-	-	-	-
13	MA	expandiere rekursive Makros	8192	1											-	-	-	-	-	-
14	CA	übersetze C-casts	16384	1											-	-	-	-	-	-
15	II	übersetze C-**`IIF`**-Strukturen	32768	1											-	-	-	-	-	-
16	QU	übersetze C-Anführungszeichen	65536	1											-	-	-	-	-	-
17	LO	übersetze C-Logik	131072	1											-	-	-	-	-	-
18	AN	deklariere **`TYPE`**s ohne C-Type als `ANY`	262144	1											-	-	-	-	-	-
19	UC	markiere unsichere **`CAST`**s	524288	0											-	-	-	-	-	-
20	MI	markiere **`#endif`**/**`#else`** bei **`#ifdef`**/**`#ifndef`**	1048576	0											-	-	-	-	-	-
21	ER	markiere Fehler als BIN Zahlen	2097152	1											-	-	-	-	-	-
22	ET	markiere Fehler in Textform	4194304												-	-	-	-	-	-
23	MB	markiere entfernte Blöcke	8388608												-	-	-	-	-	-
24	EX	markiere entfernte **`extern`** `"C"{` Zeilen	16777216												-	-	-	-	-	-
25	DT	verwende interne C-Type Umwandlungen	33554432	1											-	-	-	-	-	-
26	DF	verwende interne C-Header Ersetzungen	67108864	1											-	-	-	-	-	-

9 Anhang