

TAL To C/C++ Translator – TTC –

Nutzerdokumentation

Inhalt:

Die Nutzerdokumentation beschreibt die Anwendung des TAL To C/C++ Translators (TTC) zur Konvertierung von Programmen der Programmiersprache TAL (Transaction Application Language) optional in die Programmiersprachen C oder C++.

Produktversion: 2.5

erstellt von: pro et con Innovative Informatikanwendungen GmbH
Dittesstraße 15
09126 Chemnitz

erreichbar unter: 0371 / 270951-0

Stand: 01.12.2009

© pro et con GmbH



Änderungshistorie

Datum	Produktversion	Beschreibung	Autor
22.08.2006	2.5	Erstellung	Anja Beier (pro et con)
01.12.2009	2.5	Anpassung Kontaktdaten	Yvonne Zimmermann (pro et con)

Copyright:

Das Kopieren und Vervielfältigen der Nutzerdokumentation bzw. von Teilen davon ist nur mit ausdrücklicher Genehmigung der Firma pro et con statthaft. Alle erwähnten Programme und Namen sind Warenzeichen oder eingetragene Warenzeichen der entsprechenden Firmen.

Einschränkung der Gewährleistung:

Fehler im Inhalt dieser Nutzerdokumentation sind nicht ausgeschlossen. pro et con übernimmt keine Garantie für die Korrektheit und haftet nicht für Schäden, welche aus der Anwendung dieser Nutzerdokumentation entstehen. Da sich Fehler trotz aller Bemühungen nie vollständig vermeiden lassen, sind Hinweise und Kritiken zu deren Beseitigung jederzeit willkommen.

Kontakt:

Bei auftretenden Problemen, Fragen und Hinweisen treten Sie bitte wie folgt mit pro et con in Kontakt:
Fax: (+49 371) 270951-29
Telefon: (+49 371) 270951-0
e-mail: proetcon@proetcon.de

Dokumenthistorie

Version 1.0, Dezember 1997:

Version 1.2, Dezember 1998:

Version 1.7, Oktober 1999:

Version 2.0, Oktober 2000:

Version 2.5, August 2006:

Die Nutzerdokumentation beschreibt die Version 2.5. mit Stand August 2006.

Version 2.5 unterscheidet sich durch die folgenden Veränderungen, Verbesserungen und Erweiterungen von Version 2.0:

1. TTC in der Version 2.5 wurde nach MS Windows portiert und ist lauffähig unter Windows NT, Windows 2000, Windows XP.
2. Das Graphical User Interface *Tgo* ist nicht mehr Bestandteil des TTC

Version 2.0 unterscheidet sich durch die folgenden Veränderungen, Verbesserungen und Erweiterungen von Version 1.7:

1. Fehler bei Verwendung eines Feldnamens als Laufvariable beseitigt:

```
INT I[0:0];  
...  
FOR I := 1 TO N Do  
...
```

wurde wie folgt nach C konvertiert:

```
int i[1];  
...  
for (*i; *i!n; *i++)  
...
```

Dies wurde korrigiert in:

```
int i[1];  
...  
for (*i; *i!n; (*i)++)
```

2. Fehler bei der Konvertierung eines Literals beseitigt:

```
LITERAL -32768
```

wurde falsch konvertiert. Dieser Fehler wurde beseitigt.

3. Die Laufzeitbibliothek *ttcsys* ist sowohl als "shared library" als auch als "userlibrary" zu nutzen.

In Version 1.7 wurde nur die Nutzung als "shared library" unterstützt.

4. Fehler bei der Behandlung von leeren TAL-Blöcken wurden beseitigt:

Beispiel.:

```
BLOCK xyz;  
! LEERER BLOCK  
END BLOCK;
```

erzeugte Syntaxfehler. Dieser Fehler wurde beseitigt.

- Die Standardfunktionen `LMIN`, `LMAX`, `SPECIAL` wurden realisiert.
- In Version 2.0 ist eine (eingeschränkte) Analyse von lokalem SQL realisiert.

Beispiel.:

```
INT PROC FTXT`BEREITSTELLEN (FNR);
INT FNR;
BEGIN
INT AUSWAHL;
BEGIN DECLARE SECTION
...
END DECLARE SECTION;
```

wird korrekt analysiert. Version 1.7 erlaubte nur die Analyse von globalem SQL.

- Es existiert eine neue Translatoroption `-r <filename>`

Wird diese eingestellt, dann werden im TAL-Quelltext eingezogene `?SOURCE`-Abschnitte wie folgt im File `<filename>` dokumentiert:

```
[651]: `sphz51`[''] 1 0
[657]: `SP607U`['BLK`SYSTEM`BAUSTEINE`] 59 1
[664]: `HDM.SP607U`['SYSTEM`BAUSTEINE`] 39 1
[671]: `HDM.DS191S`[''] 63 1
[678]: `HDM.MD288S`[''] 66 1
[684]: `HDM.MD291S`[''] 70 1
[690]: `HDM.MDK79T`[''] 74 1
[696]: `HDM.DS799S`[''] 76 1
[702]: `LIB.LIBADR`['SV`GLOBAL`ADR`] 78 1
[708]: `HDM.SP607U`['BOOLEAN`LITERALS`] 81 1
[714]: `HDM.SP607U`['FILE`LITERALS`] 92 1
```

- Die Konvertierung von initialisierten Strukturen mit dem Parameter `DIRECTINITIAL_REDEFINE` wird nicht unterstützt, wenn eine Strukturkomponente eine andere überlagert.

Version 1.7 generierte hier eine falsche Initialisierung. Version 2.0 schreibt eine Warnung

```
warning G626 line %d: variable %s redefiniert und direkt initialisiert
aus.
```

- Die Initialisierung einer TAL-Struktur mit einem `INT`-Feld bei gleichzeitiger Verwendung des Konfigurationsparameters `DIRECTINITIAL` kann bei der Konvertierung zu Ausrichtungsfehlern führen. Das ist auf der TAL-Seite manuell zu prüfen. Der TTC generiert in diesem Fall eine Warnung:

```
warning G626 line %d: INT-Feld fuer direkte Strukturinitialisierung
```

- TTC realisiert Warnungen, wenn im TAL-Sourcecode Typen auftreten, deren Konvertierung nicht unterstützt wird. Die in diesem Fall bisher ausgeschriebene Warnung

```
cclass ... illegal C_STRINGCST
```

wurde ersetzt durch

```
TAL-Typ %s wird nicht behandelt
```

Version 1.7 unterscheidet sich durch die folgenden Veränderungen, Verbesserungen und Erweiterungen von Version 1.2:

1. Die Architektur der Laufzeitbibliothek `ttcsys` wurde systematisiert. Es existieren nunmehr nur noch drei h-Files
 - `rtalh`,
 - `rtvarh`,
 - `rtlh`,und die Bibliothek `rtlbo`, welche an den generierten C/C++ Code gelinkt wird. Dies kann sowohl dynamisch als auch statisch geschehen.
2. Zur Verbesserung der Wartbarkeit wurde optional eine alternative Form der Strukturinitialisierung im C/C++ Code realisiert (Einstellung über einen Parameter in der Konfigurationsdatei).
3. Das Schlüsselwort `EXT` kann gleichzeitig als Bezeichner auftreten.
4. In Version 1.7 werden bei bitweisem Zugriff auf Bezeichner auch Literale akzeptiert und korrekt konvertiert. So ist folgende Schreibweise legitim:

```
trace^flags.<tf^daily>
```
5. Die Standardfunktionen `FIX`, `FIXD` und `FIXI` wurden realisiert.
6. Die Funktionalität des Formatierungsfiles `ttc.cfg` wurde erweitert. Es existieren verbesserte Möglichkeiten, den generierten C/C++ Quelltext zu formatieren.
7. Die Heuristik für das Einfügen der TAL-Originalkommentare in den generierten C/C++-Quelltext wurde in der Version 1.7 verbessert.

Version 1.2 unterscheidet sich durch die folgenden Veränderungen, Verbesserungen und Erweiterungen von Version 1.0:

1. Eine Erweiterung der Funktionalität des Files `tal.stp` gegenüber der Version 1.0. Im File können Vorgaben (Signaturen) für Prozeduren definiert werden. Es ist damit möglich, bei der Konvertierung von TAL-Prozeduren den C-Typ der Parameter und des Rückgabewertes vorzugeben. Diese Typen werden dann bei der Konvertierung erzwungen.
2. `TAL-DEFINES` wurden in der Version 1.0 grundsätzlich expandiert. In der Version 1.2 werden `TAL-DEFINES`, welche Konstanten darstellen, nicht automatisch expandiert, sondern sie können optional in `#define`-Präprozessoranweisungen in C konvertiert werden. Die neue Option `-ept` unterscheidet die verschiedenen Konvertierungsmöglichkeiten.
3. Die Standardfunktion `OVERFLOW` wurde realisiert.
4. TTC in der Version 1.0 war lauffähig unter dem Betriebssystem HP-UX. TTC in der Version 1.2 ist nunmehr lauffähig unter den Betriebssystemen HP-UX, Sun Solaris, Windows 2000, Windows NT.
5. TTC in der Version 1.0 erzeugte C/C++ Code für TNS/C einschließlich Cfront. TTC in der Version 1.2 generiert optional C/C++ Code für:

TNS/C einschließlich Cfront,
NMC/NMCPLUS.

Die gegenüber der Version 1.0 neue Option `-native` realisiert die Einstellung auf NMC/NMCPLUS.
6. Die TAL-Compilerdirektive `SEARCH` wird in der Version 1.2 unterstützt.
7. In der Version 1.2 wird optional `char` bzw. `unsigned char` für `STRING`-Variablen generiert. Bei Einstellung der gegenüber der Version 1.0 neuen Option `-nu` wird `char` generiert.

8. TTC wurde um ein Zerteilungstool *div* erweitert.

div gliedert C/C++ Zielcode, der von TTC erzeugt wurde, automatisch in einzelne Quelltextdateien. Die Zerteilung orientiert sich an der ursprünglich im TAL-Sourcecode vorliegenden Aufteilung in Sektionen. Die gegenüber der Version 1.0 neuen Optionen `-div` und `-dir <filename>` steuern diesen Prozeß.
9. Das Produkt TTC in der Version 1.2 wurde internationalisiert.
Entsprechend des Wertes der Umgebungsvariablen `LANG` werden englische oder deutsche Meldungen generiert.
10. Die Funktionalität des Formatierungsfiles `ttc.cfg` wurde erweitert.
Es existieren verbesserte Möglichkeiten, den generierten C/C++ Quelltext zu formatieren. U.a. dient die neue Option `-fmt` dazu.
11. Die Heuristik für das Einfügen der TAL-Originalkommentare in den generierten C/C++-Quelltext wurde in der Version 1.2 verbessert.
Es existiert eine Option `-cmt`, mit der eingestellt wird, ob TAL-Kommentare in den C/C++ Code eingefügt werden oder nicht.
12. Es existieren neue Optionen `-cf`, und `-setup`.
Anstelle der vorgegebenen Files `ttc.cfg` und `tal.stp` können optional nutzerdefinierte Files verarbeitet werden.
13. Das Reengineering der zu konvertierenden TAL-Quellen und die Wartbarkeit des generierten C/C++ Codes wurde durch folgende Optionen verbessert:
 - Die Option `-ec` liefert Reengineering-Informationen zum Vorkommen von Ausdrücken und Bezeichnern (Deklaration, Definition, Nutzung).
 - Die Option `-dep` generiert Abhängigkeitsinformationen als Kommentare in den konvertierten C/C++ Code.
 - Die Option `-gi` realisiert die Initialisierung globaler Daten im C/C++ Code unmittelbar nach deren Definition.
 - Die Option `-uc` dokumentiert diejenigen Kommentare der TAL-Quelle, welche nicht in den C/C++ Code übernommen werden können.
 - Die Option `-o` optimiert die Generierung von MOVE-Anweisungen, Initialisierungen, Gruppenvergleichen und DEFINES.
14. Die Optionen `-nolink` und `-extern` unterstützen das Linken von verschiedenen, mit dem TTC aus TAL-Originalquellen generierten C/C++ Files.
15. Das Graphical User Interface *Tgo* wurde um die neuen Funktionalitäten erweitert.

Inhaltsverzeichnis

1	<i>Einführung</i>	9
1.1	Verwendete Notationen	9
1.2	Unterstützte Sprachen	10
1.3	Dokumentationen zum TTC	10
2	<i>Die Architektur des TTC</i>	11
3	<i>Hard- und Softwareanforderungen</i>	13
3.1	Lieferumfang des Translatorspaketes	13
3.2	Systemeinstellungen	15
3.2.1	Der Präprozessor <i>tpp</i>	15
3.2.2	Das Zerteilungstool <i>div</i>	16
3.2.3	Das Formatierungsprogramm <i>tal_only</i>	16
3.2.4	Der Translator <i>cgn</i>	16
3.3	Nicht realisierte Sprachkonstruktionen	17
4	<i>Der Präprozessor tpp</i>	18
4.1	Aufruf und Optionen des <i>tpp</i>	18
4.2	Die durch <i>tpp</i> generierten Files	20
4.3	Die Behandlung der TAL-Compilerdirektiven	20
5	<i>Das Formatierungsprogramm tal_only</i>	22
6	<i>Der Translator cgn</i>	23
6.1	Aufruf und Optionen des <i>cgn</i>	23
6.2	Die durch <i>cgn</i> generierten Files	26
6.3	Die Gestaltung der Meldungen	27
6.4	Die generierten Zerteilerinformationen	28
6.5	Die Behandlung der <code>INVOKE</code> -Strukturdefinitionen	28
6.5.1	Aufbau der Eingabedatei	29
6.5.2	Generierung von Dateien für die <code>INVOKE</code> -Behandlung	30
6.6	Linken von verschiedenen Modulen	31
6.6.1	Initialisierung globaler Daten	31
6.6.2	Das Werkzeug <code>itc</code>	32
6.7	Das Konvertieren von TAL-DEFINE's	32
6.8	Das Initialisieren von TAL-Strukturen	34
7	<i>Das File ttc.cfg</i>	37
8	<i>Das File tal.stp</i>	44
8.1	Funktionalität des Files <i>tal.stp</i>	44
8.2	Syntax und Semantik des Files <i>tal.stp</i>	44
	<i>Anhang A: Fehler und Warnungen tpp und tal_only</i>	53
A.1	Fehlermeldungen des <i>tpp</i>	53
A.2	Fehlermeldungen des Programmes <i>tal_only</i>	55
	<i>Anhang B: Fehler, Warnungen und Sorrrys des cgn</i>	56
	<i>Anhang C: Literaturverzeichnis</i>	70

1 Einführung

Der TAL To C/C++ Translator (TTC) ist ein Softwarewerkzeugkasten, welcher eine automatische Konvertierung von Programmen, die in der Programmiersprache TAL (Transaction Application Language) entwickelt wurden, optional in Zielcode der Programmiersprachen C oder C++ realisiert. Charakteristische Eigenschaften von TTC sind:

1. Konvertierte C-Strukturen, deren TAL-Originale aus DDL (Data Definition Language) stammen, sind auch unter C/C++ DDL-kompatibel.
2. Eine Konvertierung von eingebettetem NonStop SQL von TAL nach C wurde realisiert (einschließlich `INVOKE`).
3. Eine Abbildung der Guardian Calls von TAL nach C/C++ wurde realisiert.
4. Die TAL-Standardfunktionen wurden in C/C++ implementiert.
5. Der TTC generiert optional Zielcode für TNS/C einschließlich Cfront und NMC/NMCPLUS.
6. Der generierte C/C++ Code kann automatisch mit Hilfe eines Tools in einzelne Quelltextquellen zerteilt werden.
7. Die in TAL-Programmen enthaltenen Kommentare können optional in den konvertierten C/C++ Code eingebunden werden.
8. In einem Konfigurationsfile `ttc.cfg` läßt sich die Formatierung des zu konvertierenden C/C++ Codes entsprechend der Nutzeranforderungen deskriptiv beschreiben.
9. In einem File `tal.stp` können Vorgaben (Signaturen) für Prozeduren definiert werden. Es ist damit möglich, bei der Konvertierung von TAL-Prozeduren den C-Typ der Parameter und des Rückgabewertes der zu generierenden C-Funktionen vorzugeben. Diese Typen werden dann bei der Konvertierung erzwungen.
10. TTC ist ein internationalisiertes Produkt. Es wird die Umgebungsvariable `LANG` ausgewertet. Entsprechend dieser Belegung werden deutsche bzw. englische Meldungen generiert.
11. Das Softwarepaket ist lauffähig auf PC (Betriebssysteme Windows 2000, Windows NT, Windows XP)

1.1 Verwendete Notationen

Die folgenden Notationen wurden beim Erstellen der Nutzerdokumentation verwendet:

<code>courier</code>	Beispielprogramme, Programmnamen, Datennamen, Suffixe
<code>arial</code>	Text der Nutzerdokumentation
<i>italic</i>	Abkürzungen des Begriffsapparates
<code><courier></code>	frei wählbare Bezeichnungen
<code>[courier]</code>	optionale Angabe möglich
fett	Hervorhebungen

1.2 Unterstützte Sprachen

Die folgenden Sprachen werden im Konvertierungsprozeß unterstützt:

- TAL ab Version D20
- C optional TNS/C oder NMC
- C++ optional TNS/C einschließlich Cfront oder NMCPLUS
- NonStop SQL C30 für TNS/C und NMC
- DDL DDL D20

Bei der Konvertierung ist folgendes zu beachten:

Die C++-Compiler TNS/C einschließlich Cfront bzw. NMCPLUS (Stand Oktober 1999) unterstützen kein NonStop SQL. Werden TAL-Programme mit eingebettetem NonStop SQL nach C++ konvertiert, so werden die SQL-Anweisungen auskommentiert. Die konvertierten C++ Programme sind damit compiler- und linkfähig, realisieren aber bei Ihrer Ausführung nicht die korrekte Programmfunktionalität.

1.3 Dokumentationen zum TTC

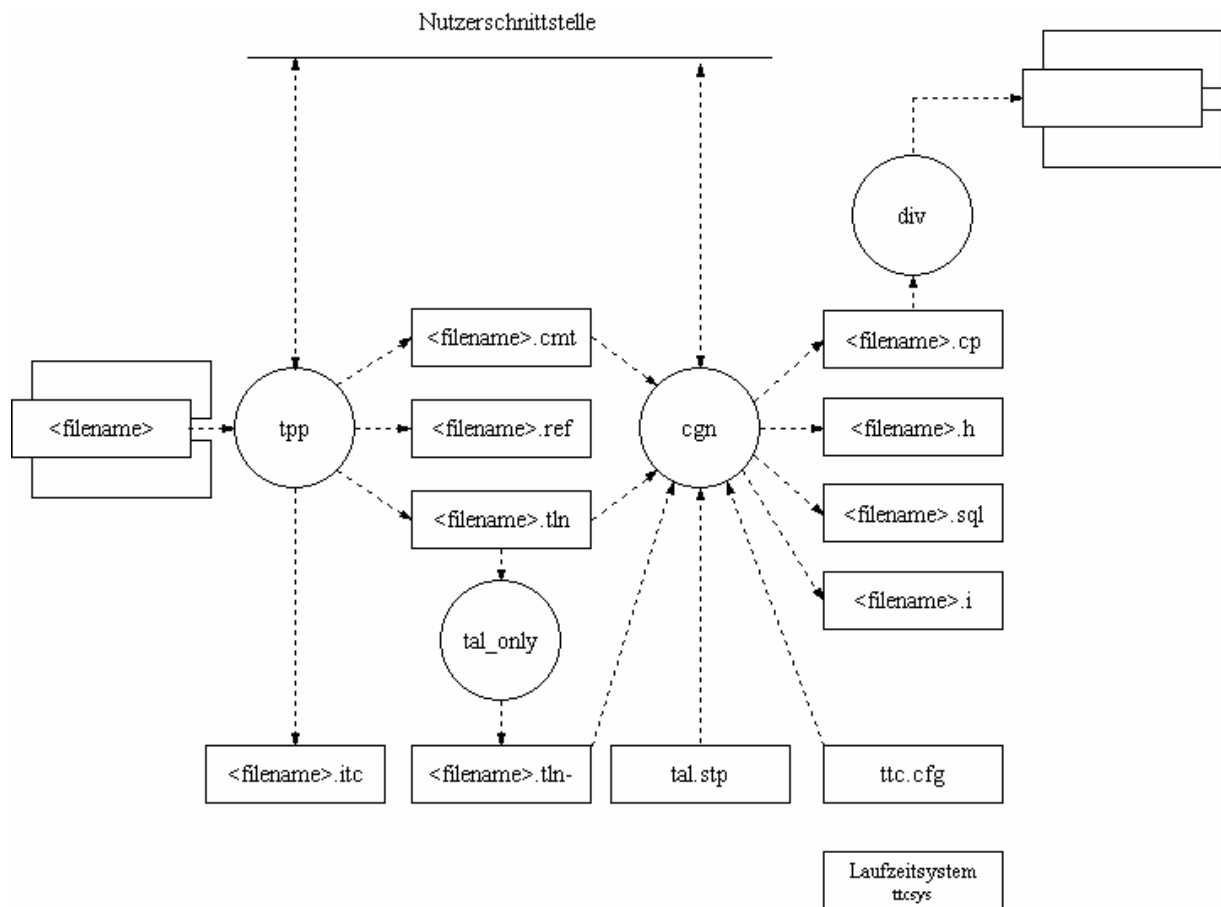
Die Dokumentationen zum TTC umfassen drei Dokumente:

1. Die Abbildungsdokumentation TAL To C/C++ [2]:
Diese beschreibt und motiviert die Abbildung von TAL-Sprachkonstruktionen in C/C++-Sprachkonstruktionen. Sie versetzt den Anwender in die Lage, mit Kenntnis des originalen TAL- Programmes das durch TTC generierte C/C++ Programm zu verstehen. Dies bildet die Voraussetzung für dessen Wartung und Weiterentwicklung.
2. Die Nutzerdokumentation zum Zerteiler *div* [1]:
Sie versetzt den Anwender in die Lage, das vom TTC generierte C/C++-Programm automatisch in einzelne Quelltextdateien aufzuteilen.
3. Die vorliegende Nutzerdokumentation zum TTC :
Sie versetzt den Anwender in die Lage, die verschiedenen Komponenten des TTC so zu handhaben, daß aus originalen TAL-Quellen ein vollständiges, fehlerfrei compilier- und linkbares C/C++-Programm konvertiert werden kann. Die Nutzerdokumentation gliedert sich wie folgt:
 - Die Abschnitte 1 bis 8 beschreiben die Arbeitsweise des TTC und dienen der Einarbeitung.
 - Anhang A und Anhang B dokumentieren die Fehlermeldungen des TTC und dienen als Nachschlagewerk bei der praktischen Arbeit mit dem Translator.

2 Die Architektur des TTC

Das nachfolgende Bild dokumentiert die Architektur des TTC. Dabei gilt:

- Kreise kennzeichnen ausführbare Programme, in diesem Fall die Komponenten, aus denen sich TTC zusammensetzt.
- Rechtecke kennzeichnen Files und Bibliotheken, welche entweder während des Konvertierungsprozesses geschrieben werden oder im Lieferumfang des Translatorpaketes enthalten sind.
- Pfeile kennzeichnen den Informationsfluß zwischen Programmen und Files.



TTC besteht aus folgenden Komponenten:

1. Dem Präprozessor *tpp*: *tpp* liest ein TAL-Quelltextfile `<filename>`, welches die `MAIN`-Prozedur des zu konvertierenden TAL-Programmes enthält und erzeugt unter Auswertung der im TAL-Quelltext stehenden Compilerdirektiven aus verschiedenen Quelltextmoduln ein vollständiges TAL-Programm `<filename>.tln`. Gleichzeitig werden die Files `<filename>.cmt` (beinhaltet die Kommentare des originalen TAL-Quellprogrammes) und `<filename>.ref` (beinhaltet Informationen, aus welchen Sektionen sich das Quelltextfile zusammensetzt) geschrieben. *tpp* wird im Abschnitt 4 dokumentiert.
2. Dem Formatierungsprogramm *tal_only*: *tal_only* liest das durch *tpp* erzeugte TAL-Programm `<filename>.tln` und bereitet es in einem File `<filename>.tln~` so auf, daß es mittels eines Editors durch den Nutzer betrachtet werden kann. *tal_only* wird im Abschnitt 5 dokumentiert.
3. Dem Translator *cgn*: *cgn* liest das TAL-Programm (`<filename>.tln` bzw. `<filename>.tln~`) und generiert optional C- oder C++-Code in den Files `<filename>cp` und `<filename>h`. Parallel dazu können entsprechend eingestellter Optionen die Files `<filename>.sql` (beinhaltet Informationen zu den im TAL-Quelltext auftretenden SQL-Anweisungen) und `<filename>.i` (beinhaltet den TAL-Quelltext nach Auflösung der `DEFINE`-Anweisungen) geschrieben werden. *cgn* wird im Abschnitt 0 dokumentiert.
4. Dem Zerteilungstool *div*, welches aus TAL konvertierte, vollständige C/C++ Programme entsprechend dem Layout des zu konvertierenden TAL-Programmes in einzelne Quelltextmodule aufteilt. *div* wird im "Nutzerhandbuch Zerteiler" [1] beschrieben.
5. Dem Laufzeitsystem *ttsys*, welches aus einer Sammlung von Headerfiles und Bibliotheken besteht. Die Bibliotheken sind auf dem Zielrechner zu installieren und mit dem konvertierten C/C++ Programm zum ausführbaren Programm zu linken. Dies garantiert die korrekte Funktionalität der generierten C/C++-Programme auf der Zielarchitektur. *ttsys* wird im Abschnitt 3.1 dokumentiert.
6. Dem File `tal.stp`, welches in formalisierter Form eine Beschreibung der Standardfunktionen und Guardian Calls enthält. Darüberhinaus können in diesem File Vorgaben (Signaturen) für Prozeduren definiert werden. Es ist damit möglich, bei der Konvertierung von TAL-Prozeduren den C-Typ der Parameter und des Rückgabewertes vorzugeben. Diese Typen werden dann bei der Konvertierung erzwungen. `tal.stp` wird im Abschnitt 8 dokumentiert.
7. Dem File `ttc.cfg`, welches zur Beschreibung der Formatierung des zu generierenden C/C++-Zielcodes genutzt wird. `ttc.cfg` wird im Abschnitt 7 dokumentiert.

tpp, *tal_only*, *div* und *cgn* sind als kommandozeilenorientierte Werkzeuge nutzbar.

3 Hard- und Softwareanforderungen

3.1 Lieferumfang des Translatorspaketes

Der Lieferumfang des Translatorspaketes wird unterteilt in logisch zusammenhängende Pakete:

- Komponenten des Translators,
- Komponenten des Laufzeitsystems *ttcsys*.

Der Translator besteht aus folgenden Programmen und Files:

Komponenten des Translators:

Name	Bedeutung
tp	Der eigentliche Präprozessor.
tal-only	Ein Formatierungsprogramm, welches aus einem durch den Präprozessor erzeugten File sämtliche redundante Informationen entfernt.
cgn	Der eigentliche Translator.
div	Liest das konvertierte C/C++ Programm einschließlich der generierten Zerteilungsinformationen ein und generiert daraus mehrere C/C++ Sourcen.
ttc.cfg	Ein Formatierungsfile zur Beschreibung des zu generierenden Codes.
tal.stp	Ein File mit der Beschreibung aller Standardfunktionen und Guardian Calls.
ivkdefs.awk	Ein Skript für <i>awk</i> , <i>gawk</i> oder <i>nawk</i> zur Erzeugung von Index- und Strukturdateien zur Aufbereitung von <i>INVOKE</i> -Strukturen.
sqlstruct.ivk	Eine Datei mit Beschreibungen der Strukturen <i>SQLSA</i> und <i>SQLCA</i> .
itc	Ein Programm für Initialisierungsfunktionen.

Folgende Programme und Files sind für die Arbeit des Laufzeitsystems *ttcsys* notwendig:

Komponenten von *ttcsys* :

Name	Bedeutung
rtvarh	Headerfile, welches Deklaration und Definition von Variablen des Laufzeitsystems enthält.
rtalh	Headerfile, welches die Schnittstelle zur Laufzeitbibliothek und zum Betriebssystem darstellt. Dieses File wird direkt in den generierten C/C++ Quellcode integriert.
rtlh	Headerfile, welches Definitionen der in C/C++ implementierten Standardfunktionen enthält.
rtsyscp	Das File enthält die Implementierungen der Laufzeitroutinen.
rtlcp	Das File enthält die Implementierungen der Standardfunktionen.
rtlibo	Die Laufzeitbibliothek, die an den entstandenen C/C++-Code gelinkt werden muß. Die entsteht aus den Files <i>rtsyscp</i> und <i>rtlcp</i> . Sie kann vom Anwender eigenständig erstellt werden. Damit hat der Anwender volle Kontrolle über die Art des Linkens der Bibliothek (statisch oder dynamisch).

Die Bedeutung der einzelnen Files und Programme wird in den entsprechenden Abschnitten der Nutzerdokumentation erläutert.

3.2 Systemeinstellungen

Für alle Komponenten des Translatorpaketes gilt:

1. Sie sind lauffähig auf:
PC, Betriebssystem Windows 2000, Windows NT, Windows XP.
2. Die nachfolgend beschriebenen Systemeinstellungen im kommandoorientierten Modus werden für Windows-Betriebssysteme angegeben.
3. Alle Komponenten des TTC werten die eingestellte Sprachumgebung aus. Ist eine deutsche Sprachumgebung definiert (also die Umgebungsvariable `LANG` auf `de` gesetzt), so erscheinen alle Ausschriften, Meldungen, Menüpunkte, ... in deutscher Sprache. In allen anderen Fällen (auch, wenn `LANG` nicht gesetzt ist) erscheinen sie Englisch. Im weiteren wird immer auf die deutschsprachige Version Bezug genommen.

3.2.1 Der Präprozessor *tpp*

Wird *tpp* als kommandoorientiertes Werkzeug unabhängig von der graphischen Benutzeroberfläche genutzt, so sind die folgenden Systemeinstellungen notwendig:

1. Den Suchpfad auf das Verzeichnis setzen, in welchem sich *tpp* befindet.

Beispiel.:

Angenommen, *tpp* befindet sich im Verzeichnis `/usr/local/translator/bin`, so wird der Suchpfad mit der Bourne Again-Shell (*bash*) wie folgt gesetzt:

```
set PATH $PATH:/usr/local/translator/bin
```

2. Setzen der Umgebungsvariable `TALSRC` auf eine Verzeichnispfadliste. Diese Verzeichnisse werden von *tpp* nach TAL-Quelltexten durchsucht, die in das zu bearbeitende TAL-Programm eingezogen werden sollen.

Beispiel.:

Angenommen, *tpp* soll in den Verzeichnissen `/usr/local/translator/src1` und `/usr/local/translator/src2` nach TAL-Quellen suchen. Dann wird `TALSRC` mit der Bourne Again-Shell (*bash*) wie folgt gesetzt:

```
export TALSRC = "/usr/local/translator/src1"  
export TALSRC = $TALSRC:"/usr/local/translator/src2"
```

Die Umgebungsvariable muß nur gesetzt werden, wenn Quelltexte über diesen Mechanismus eingezogen werden sollen. Die unterschiedlichen Möglichkeiten des Einziehens von Quelltexten beschreibt Abschnitt 4.

3.2.2 Das Zerteilungstool *div*

Für *div* sind die folgenden Systemeinstellungen notwendig:

Den Suchpfad auf das Verzeichnis setzen, in welchem sich *div* befindet.

Beispiel.:

Angenommen, *div* befindet sich im Verzeichnis `/usr/local/translator/bin`, so wird der Suchpfad mit der Bourne Again-Shell (*bash*) wie folgt gesetzt:

```
set PATH $PATH:/usr/local/translator/bin
```

3.2.3 Das Formatierungsprogramm *tal_only*

Für *tal_only* ist die folgende Systemeinstellung notwendig:

Den Suchpfad auf das Verzeichnis setzen, in welchem sich *tal_only* befindet.

Beispiel.:

Angenommen, *tal_only* befindet sich im Verzeichnis `/usr/local/translator/bin`, so wird der Suchpfad mit der Bourne Again-Shell (*bash*) wie folgt gesetzt:

```
set PATH $PATH:/usr/local/translator/bin
```

3.2.4 Der Translator *cgn*

Für *cgn* sind die folgenden Systemeinstellungen notwendig:

1. Den Suchpfad auf das Verzeichnis setzen, in welchem sich *cgn* befindet.

Beispiel.:

Angenommen, *cgn* befindet sich im Verzeichnis `/usr/local/translator/bin`, so wird der Suchpfad mit der Bourne Again-Shell (*bash*) wie folgt gesetzt:

```
set PATH $PATH:/usr/local/translator/bin
```

2. Setzen der Umgebungsvariable `TAL_SETUP` auf ein Verzeichnis, in welchem sich die Datei `tal.stp` befindet. Diese Datei beinhaltet die Beschreibung der Standardfunktionen und Guardian Calls.

Beispiel.:

Angenommen, `tal.stp` steht im Verzeichnis `/usr/local/translator/bin`, dann wird `TAL_SETUP` mit der Bourne Again-Shell (*bash*) wie folgt gesetzt:

```
set TAL_ETUP /usr/local/translator/bin
```

3.3 Nicht realisierte Sprachkonstruktionen

Einige TAL-Sprachkonstruktionen sind in der vorliegenden Version des TTC nicht realisiert. Diese sind:

1. ASSERT-Anweisung
2. CODE-Anweisung
3. DROP-Anweisung
4. STORE-Anweisung
5. STACK-Anweisung
6. USE-Anweisung
7. Standardfunktionen:
 - \$AXADR (wird nicht realisiert)
 - \$CARRY (wird nicht realisiert)
 - \$BITLENGTH
 - \$BITOFFSET
 - \$BOUNDS (wird nicht realisiert)
 - \$EFLT
 - f\$EFLTR
 - \$FIXL
 - \$FIXR
 - \$FLT
 - \$FLTR
 - \$IFIX
 - \$LFIX
 - \$OVERFLOW (wird nicht realisiert)
 - f\$READCLOCK
 - \$RP (wird nicht realisiert)
 - \$SWITCHES (wird nicht realisiert)
 - \$USERCODE (wird nicht realisiert)
8. speicherplatzlose Felder
9. nicht zur Konvertierungszeit berechenbare Ausdrücke in Initialisierungslisten und Feldgrößen
10. Typ UNSIGNED
11. Bezugnahme auf 'G', 'L' und 'S' Register
12. Speicherkonsistente Abbildung, also eine definierte Lage von unabhängigen Daten zueinander und Bezugnahme aus daraus resultierenden Adreßrechnungen

4 Der Präprozessor *tpp*

Jedes TAL-Programm enthält Compilerdirektiven und Kommentare. Es wird deshalb in einem ersten Schritt innerhalb der Konvertierung von TAL nach C/C++ mit einem Präprozessor bearbeitet. Der Präprozessor realisiert folgende Aufgaben:

1. Auswertung und Behandlung von TAL-Compilerdirektiven (siehe Abschnitt 4.3),
2. Sammeln von Originalkommentaren zur späteren Nutzung in generierten C/C++-Quelltexten,
3. Ersetzen von Kommentaren und Compilerdirektiven durch Zwischencodedirektiven, welche einen Rückbezug zu Originalquellen ermöglichen (siehe Abschnitt 4.2).

tpp liest ein File mit dem Namen `<filename>`, welches die MAIN-Prozedur des zu konvertierenden TAL- Quellprogrammes beinhaltet und liefert ein TAL-Programm, welches keine TAL-Direktiven und Kommentare mehr enthält.

Das erzeugte File besitzt den Namen `<filename>.tln`.

Dieses File wird als Eingabe für weitere Konvertierungsphasen genutzt.

Im File `<filename>.tln` wurden an den Stellen, an denen im Originalfile Kommentare oder Direktiven standen, spezielle Zwischencodedirektiven eingefügt. Diese dienen späteren Konvertierungsphasen zum Rückbezug auf korrekte Positionen in den originalen Quelltextfiles.

Originalkommentare werden in einem File `<filename>.cmt` gesammelt.

In diesem Kapitel wird:

1. Der Aufruf von *tpp* einschließlich der entsprechenden Optionen dokumentiert.
2. Der Aufbau und die Bedeutung der entstehenden Files beschrieben.
3. Auf die Auswertung der einzelnen TAL-Compilerdirektiven durch *tpp* eingegangen.

4.1 Aufruf und Optionen des *tpp*

tpp wird mit dem folgenden Kommando gestartet:

```
tpp <option 1> ... <option n> <filename>
```

`<filename>`:

ist der Name eines TAL-Programmes, welches mit dem Präprozessor bearbeitet werden soll. Es muß nicht im aktuellen Verzeichnis stehen (Vgl. die Optionen `-I` und `-r`). Das Programm enthält die MAIN-Funktion.

`<option 1> ... <option n>`:

steuern den Präprozessorlauf. Die Angabe der folgenden Optionen ist möglich:

Präprozessoroptionen:

Option	Bedeutung
-I <path-list>	Realisierung der TAL-Compilerdirektiven <code>SOURCE</code> und <code>SECTION</code> . Verzeichnisse, die zusätzlich durchsucht werden sollen, sind mit dieser Option anzugeben. Die Angabe mehrerer absoluter oder relativer Pfadnamen, die durch Komma (,) getrennt werden, ist möglich. Der Quelltext des zu konvertierenden Programmes und der mit <code>SOURCE</code> eingezogenen Files und Sektionen wird in diesen Verzeichnissen gesucht.
-r <path>	Realisierung der TAL-Compilerdirektiven <code>SOURCE</code> und <code>SECTION</code> . Unter dem Verzeichnis <path> werden rekursiv alle Unterverzeichnisse nach TAL-Quelltexten durchsucht. Ist diese Option nicht angegeben, so wird das aktuelle Verzeichnis verwendet.
-l <filename>	Behandlung der TAL-Compilerdirektive <code>SEARCH</code> . Die Filenamen, welche bei <code>SEARCH</code> vorkommen, werden in ein File <filename> gespeichert, um von <i>itc</i> (Vgl. Abschnitt 32) bearbeitet zu werden.
-D <toggle-name-list>	Realisierung der TAL-Compilerdirektiven <code>SETTOG</code> und <code>RESETDOC</code> . Toggle-Informationen müssen dem Präprozessor bekannt sein. Dies geschieht mit folgendem Mechanismus: Die Toggles werden in der Liste <toggle-name-list> durch Doppelpunkte (:) getrennt angegeben. Sie müssen der TAL-Syntax für Toggles genügen.
-H	Ausgabe der Namen der einzelnen, mit <code>SOURCE</code> eingezogenen Files oder Sektionen auf Standardausgabe. Entsprechende Einrückungen erleichtern das Erkennen, welches File welches andere File bzw. welche Sektion einzieht. Es werden vollständige Pfadnamen angegeben.
-o <filename>	Standardmäßig erzeugt der Präprozessor <i>tpp</i> ein Ausgabefile <filename>.tln, wobei <filename> der Name des zu konvertierenden TAL-Files ist. Mit dieser Option kann der Name des Ausgabefiles vom Nutzer festgelegt werden.
-c <filename>	Standardmäßig erzeugt der Präprozessor <i>tpp</i> ein Kommentarfile <filename>.cmt, wobei <filename> der Name des zu konvertierenden TAL-Files ist. Mit dieser Option kann der Name des Kommentarfiles vom Nutzer festgelegt werden.

Generell muß nach Angabe der Option (-r, -I, -D, -H, -o oder -c) wenigstens ein Leerzeichen stehen. Innerhalb der Optionsparameter (Listen und Filenamen) dürfen keine Leerzeichen oder Tabulatoren vor- kommen.

Die Optionen -D und -I können mehrfach in der Kommandozeile angegeben werden. Zusätzlich zu den Optionen -r und -I kann auch noch die Umgebungsvariable `TALSRC` zur Realisierung der `SOURCE`- und `SECTION`-Direktiven genutzt werden. Der Wert dieser Umgebungsvariablen kann eine Pfadliste (wie bei -I) sein.

Zur Suchreihenfolge ist folgendes festgelegt:

- Zuerst werden die bei -I angegebenen Pfade in der angegebenen Reihenfolge durchsucht.
- Danach werden alle in der Umgebungsvariablen `TALSRC` angegebenen Pfade in der angegebenen Reihenfolge durchsucht.
- Zuletzt werden alle Verzeichnisse, die unter dem bei -r angegebenen Pfad stehen (einschließlich aller Unterverzeichnisse) durchsucht. Eine Reihenfolge, in welcher diese durchmustert werden, ist dabei nicht festgelegt.

Die Suchreihenfolge muß dann beachtet werden, wenn in den angegebenen Verzeichnissen mehrere Files mit dem gleichen Namen vorkommen. Es wird mit dem ersten gefundenen Quelltextfile des entsprechenden Namens gearbeitet.

4.2 Die durch *tpp* generierten Files

Beim Aufruf des *tpp* mit `<filename>` als Argument werden die folgenden Files im aktuellen Verzeichnis erzeugt:

1. `<filename>.tln`

Dieses File beinhaltet den vollständigen TAL-Quelltext, der konvertiert werden soll. Alle durch TAL-Compilerdirektiven adressierten Sektionen und Files wurden eingezogen. Desweiteren wurden Originalkommentare und die TAL-Compilerdirektiven selbst entfernt. Um die Position eines TAL-Konstrukts bezüglich der originalen TAL-Quellen beim Konvertieren zur Verfügung zu haben, werden folgende Zwischencodedirektiven eingefügt:

- FILE[`<name>`]
- LINE[`<zeilennummer>`,`<fileoffset>`]
- COL[`<spaltennummer>`,`<fileoffset>`]
- ENDFILE
- COMMENT[`<fileoffset>`,`<laenge>`]

Das Verständnis dieser Zwischencodedirektiven ist für die Translatornutzung nicht von Belang. Auf eine detaillierte Erläuterung wird deshalb verzichtet. Wichtig ist, daß das `.tln`-File nicht mit einem Editor manipuliert werden darf, da es als Eingabe für den weiteren Konvertierungsprozeß dient.

2. `<filename>.cmt`

Alle im Originaltext vorhandenen Kommentare werden in diesem File aufgehoben. Aufgrund seines Formates ist dieses File nicht mittels Editoren lesbar. Es bildet die Basis für weitere Werkzeuge, welche das Einfügen von Originalkommentaren in den zu generierenden C/C++-Code unterstützen.

3. `<filename>.ref`

Dieses File beinhaltet Informationen darüber, aus welchen Sektionen sich das Quelltextfile zusammensetzt und wo genau (von Zeile/Spalte bis Zeile/Spalte) diese im File stehen stehen. Außerdem wird festgehalten, welche Sektionen oder Files an welchen Stellen (Zeile/Spalte) mittels `SOURCE` eingezogen werden. Das `.ref`-File liefert somit für den Anwender nützliche Informationen darüber, aus welchen Komponenten sich das zu konvertierende TAL-Programm zusammensetzt. Im Konvertierungsprozeß selbst wird es durch die Werkzeuge nicht weiter ausgewertet.

4.3 Die Behandlung der TAL-Compilerdirektiven

Compilerdirektiven in TAL sind durch ein Fragezeichen (?) in der ersten Quelltextspalte gekennzeichnet.

Der Präprozessor erkennt alle Compilerdirektiven der Sprache TAL syntaktisch und protokolliert auch Syntaxfehler in den Direktiven. Im weiteren werden die Direktiven dokumentiert, die vom *tpp* nicht nur analysiert, sondern auch verarbeitet werden. Alle anderen werden aus dem Quelltext gestrichen und lösen keine Aktionen aus.

1. `SOURCE`

Wenn keine Liste mit Sektionsnamen angegeben wurde, wird das gesamte File, dessen Name hinter `SOURCE` angegeben wurde, eingezogen. Ist diese Liste vorhanden, werden nur die entsprechenden Sektionen eingezogen. Die Reihenfolge, in der sie eingezogen werden, ist die, in der die Sektionen auch im Quelltextfile stehen. Diese kann von der Reihenfolge in der

Sektionsliste abweichen. Der eingezogene Quelltext kann seinerseits auch wieder Compilerdirektiven enthalten, die analog behandelt werden. Ist das File oder die Sektion nicht vorhanden, wird eine Fehlermeldung ausgegeben.

2. SECTION

Diese Direktive kennzeichnet den Beginn einer neuen Sektion in diesem File. Wird mit SOURCE eine Liste von Sektionen aus einem File eingezogen, so wird der zwischen zwei SECTION-Direktiven stehende Quelltext nur eingezogen, wenn der Name der ersten SECTION-Direktive (Bezeichner unmittelbar hinter SECTION) in der Liste vorhanden ist. Eine Sektion wird entweder durch das Fileende oder durch die nächste SECTION-Direktive beendet.

3. SEARCH

Die auf das Schlüsselwort SEARCH folgenden Bezeichner werden optional in einem File gesammelt, welches mit *itc* (Vgl. 6.6.2) weiterverarbeitet wird. Dies vereinfacht das Linken. SEARCH und dessen Parameter werden durch *tpp* aus dem Quelltext gestrichen.

4. SETTOG

Mit dieser Direktive werden ein oder mehrere Toggles gesetzt. Toggles sind Schalter, deren Wert bei der IF/IFNOT-Direktive ausgewertet wird. Ein Toggle kann ein Bezeichner oder auch eine vorzeichenlose ganze Zahl sein. Werden mehrere Toggles in einer Direktive gesetzt, so müssen sie durch Kommata getrennt werden.

5. RESETTOG

Mit dieser Direktive werden Toggles zurückgesetzt.

6. IF/IFNOT und ENDIF

Mit diesen Direktiven wird eine bedingte Übersetzung realisiert. Nach IF/ENDIF ist ein Toggle (Name oder Nummer) anzugeben. Wurde das Toggle vorher im Präprozessorlauf mittels SETTOG gesetzt, so wird der nach einer IF-Direktive stehende Quelltext eingezogen und der nach einer IFNOT-Direktive stehende Text nicht. Wurde das Toggle nicht gesetzt (Anfangszustand) oder mit RESETTOG zurückgesetzt, so wird umgekehrt der nach der IFNOT-Direktive stehende Quelltext eingezogen und der nach einer IF-Direktive stehende Text nicht. Eine IF/IFNOT-Direktive muß mit einer ENDIF-Direktive abgeschlossen werden. Fehlt diese, wird eine Fehlermeldung generiert. Es ist möglich, IF/IFNOT-Direktiven zu schachteln.

5 Das Formatierungsprogramm *tal_only*

Die im File `<filename>.tln` enthaltenen ZwischencodDirektiven erschweren dessen Lesbarkeit. Das File dient als Eingabe für den weiteren Konvertierungsprozeß und ist zum Betrachten durch den Anwender ungeeignet. Es ist jedoch nützlich, das zu konvertierende TAL-Programm in seiner Gesamtheit betrachten zu können.

Zu diesem Zweck wird das Formatierungs-Programm *tal_only* bereitgestellt. Es erwartet den Namen eines Files mit dem Suffix `.tln` als Parameter und erzeugt daraus ein neues File `<filename>.tln~`. Dieses enthält den TAL-Quelltext aus dem `.tln`-File ohne ZwischencodDirektiven und kann mit einem Texteditor betrachtet werden.

Auch dieses File kann als Eingabe für den weiteren Konvertierungsprozeß dienen. In diesem Fall werden aber alle Quelltextpositionen in Fehlermeldungen, Warnungen und Reengineering-Informationen bezüglich dieses Files und nicht mehr bezüglich der Original-Quellen angegeben. Auch die Informationen zu den Kommentaren sind nicht mehr vorhanden. Zu Testzwecken ist es dennoch nützlich. Es ist einfacher, im Konvertierungsprozeß aufgetretene Fehler, Warnungen und Sorrys (Siehe Anhang B) in einem File zu lokalisieren und nicht in vielen, über mehrere Verzeichnisse verteilten Quelltexten.

6 Der Translator *cg*n

6.1 Aufruf und Optionen des *cg*n

Der Translator *cg*n ist mit dem Aufruf

```
cg n <option_1> ... <option_n> <filename>.tln
```

bzw.

```
cg n <option_1> ... <option_n> <filename>.tln~
```

zu starten.

<filename>.tln

ist das zu konvertierende TAL-Programm, welches durch den Präprozessor *tpp* erzeugt wurde.

<filename>.tln~

ist das zu konvertierende TAL-Programm, welches zusätzlich zu *tpp* noch mit *tal_only* bearbeitet wurde.

<option_1> ... <option_n>

sind die zur Verfügung stehenden Optionen, sie werden im weiteren dokumentiert.

Translatoroptionen:

Option	Bedeutung
-bit	Es werden alle Zeilen in den Quelltexten auf der Standardausgabe protokolliert, in denen der Bit-Extractions-Operator (<>) verwendet wird.
-C	Es wird C-Code statt C++-Code generiert.
-cf <filename>	Kommentare aus dem File <filename> einfügen. Standardmäßig wird aus dem .cmt-File eingefügt.
-CFG <filename>	Die Formatierungsfestlegungen bezüglich des Zielcodes werden diesem File entnommen. Im default-Fall werden die Formatierungsfestlegungen einem File <i>ttc.cfg</i> im aktuellen Verzeichnis entnommen. Existiert dies nicht, treten Standardanwendungen in Kraft. Die Formatierungsfestlegungen einschließlich der Standardanwendungen sind im Abschnitt 37 erläutert.
-cmt	Originalkommentare aus dem TAL-Quellprogramm in den generierten C/C++-Code einfügen.
-dep	Generieren von Abhängigkeitsinformationen als Kommentare in das generierte C/C++-File.
-dir <filename>	In <filename> steht eine Liste von globalen Objekten (Variablendeklarationen und -definitionen, Funktionsdeklarationen, etc.), die bei der Zerteilung des entstandenen C/C++-Codes in einzelne Files im Hauptfile verbleiben und nicht in einzelne Module gezogen werden sollen. Der Aufbau von <filename> wird im Abschnitt 28 erläutert.
-div	Zerteilungsinformationen in Form von Kommentaren werden vor und nach jeder globalen Deklaration und Definition generiert. Nur, wenn ein TAL-Programm mit dieser Option konvertiert wurde, kann es anschließend mit dem Zerteilungswerkzeug zerteilt werden.

-dro	Unterdrückt die Umordnung von Daten in einer Deklaration im Zielcode.
-eb	Daten, die in BLOCK-Statements in TAL eingeschlossen sind, werden mit der Speicherklasse extern in C/C++ generiert, sofern der Block nicht das Attribut PRIVATE besitzt. Werden mehrere Module einzeln konvertiert und dann mit dem C/C++-Compiler übersetzt und gelinkt, so muß diese Option für alle Module bis auf einen angegeben werden. Dann nutzen die einzelnen Module dieselben Daten und es kommt nicht zu Linker-Fehlern.
-ec <filename>	Erzeugt ein File mit dem vorgegebenen Namen, in dem zusätzliche Informationen zum Vorkommen von Ausdrücken und Bezeichnern (Deklaration/Definition und Nutzung) bereitgestellt werden, dient zum Reengineering.
-ept	Ist diese Option nicht angegeben, werden TAL-DEFINES, welche Konstanten darstellen, in C-#define's übersetzt. Die Form der Übersetzung wird im Abschnitt 32 beschrieben. Ist diese Option angegeben, wird die Erzeugung von #defines generell verhindert. Alle TAL-DEFINES werden expandiert.
-fmt <filename>	Angabe eines zusätzlichen Konfigurationsfiles, welches Informationen zu Zeilenumbrüchen in Parameterlisten von Funktionen enthält. Das Format ist das eines allgemeinen Konfigurationsfiles bei Angabe der Option -CFG. Die Angaben in diesem File haben gegenüber den Konfigurationsoptionen wie BREAK-CALL-LEN die höhere Priorität.
-gi	Wird diese Option angegeben, so erfolgen Initialisierungen globaler Daten auch in C immer unmittelbar nach deren Definition. Sollte der Initialisierungsausdruck nicht zur Übersetzungszeit vom C-Compiler berechnet werden können, da er z.B. Variablen enthält, führt die Angabe dieser Option zu nicht-übersetzbarem C-Code, der erst noch von Hand nachbearbeitet werden muß. Ist die Option nicht gesetzt, so erfolgen die Initialisierungen globaler Daten in der Funktion Ar-init(), die am Ende des erzeugten Codes zu finden ist. Diese Option verbessert die Wartbarkeit des generierten C/C++-Codes.
-h	Es wird eine Kurzhilfe zu den Optionen über die Standardausgabe ausgegeben.
-help	Es wird die ausführliche Hilfe zu den Optionen über der Standardausgabe angezeigt. Der Option folgt eine Zeichenkette, die als Name der zu beschreibenden Option gilt. Wird hier all angegeben, erfolgt die Ausgabe der Hilfetexte zu allen Optionen.
-ic	Im TAL-Quelltext vorkommende INVOKE-Strukturen werden als Kommentare in den generierten C/C++-Code eingefügt.
-im	Generierung der Zeilennummer im Fehlerprotokoll bezogen auf die Zeilennummerierung im File <filename>.tln bzw. <filename>.tln~. Standard ist die Ausgabe der Zeilennummern bezogen auf die Originalfiles. Diese werden den Zwischencode- Direktiven des Files <filename>.tln entnommen. Als File wird dann in jedem Fall der Name des Files angegeben, welches als zu parsendes File angegeben wurde. Ohne im-Option wird in jedem Fall der Name des durch Auflösung der Include-Hierarchie aktuellen Files angegeben.

<code>-ivk <val1> <val2></code>	Behandlung von <code>INVOKE</code> -Definitionen. Die Option erwartet zwei Parameter, zunächst den Namen des Files <code><val1></code> , in dem die Indexinformationen zu den <code>INVOKE</code> -Strukturen abgespeichert sind, als zweites <code><val2></code> , die Datei, in der die Strukturdefinitionen selbst enthalten sind (Erläuterungen zu <code>INVOKE</code> siehe Abschnitt 28).
<code>-N <val></code>	Festlegung, nach wievielen Spalten <code><val></code> während der Generierung einer Zeile umgebrochen werden soll. Zeilenumbrüche erfolgen nicht in Zeichenketten und frühestens dann, wenn ein Whitespace geschrieben wird. Aus diesem Grund kann eine Zeile im Zielcode länger als <code><val></code> werden.
<code>-native</code>	Die Angabe dieser Option bewirkt, daß Code für den NMC von Tandem erzeugt wird, wenn gleichzeitig die Option <code>-tg tandem</code> gesetzt ist. Wird eine andere Zielplattform gewählt (Windows oder Unix), so hat <code>-native</code> keine Auswirkungen. Der einzige Unterschied zwischen NMC- und TNS/C-Code besteht darin, daß bei NMC-Code kein <code>-lowmem</code> -Modifikator für Daten, die in TAL im unteren Speicherbereich liegen und in C normalerweise nicht, erzeugt wird. Hinweis: Wird Code für NMC generiert, so ist eine Umwandlung von Adressen nach <code>short</code> (die bei der Übersetzung mit <code>cgn</code> entsteht) generell nicht mehr möglich und führt zu Laufzeitfehlern. Die entsprechenden Stellen sind anzupassen. Bei Funktionsparametern kann dazu die Möglichkeit der Vorgabe der Funktionssignatur im File <code>tal.stp</code> verwendet werden. Andere Stellen sind manuell im C-Code zu ändern.
<code>-nolink</code>	Es wird für Files, die eine <code>MAIN</code> -Funktion besitzen, die Generierung des Rufs der Funktion <code>-Ar-init-submodule</code> unterdrückt.
<code>-nu</code>	Anstelle des Typs <code>unsigned char</code> für <code>STRING</code> Variablen wird der Typ <code>char</code> generiert.
<code>-O <val></code>	Optimierungen für Initialisierungen, <code>MOVE</code> -Anweisungen, Gruppenvergleiche und <code>TAL-DEFINES</code> . Für <code><val></code> gilt: 0 : keine Optimierung, 1 : nur Initialisierungstabellen optimieren, 2 : nur <code>MOVE</code> -Anweisungen und Gruppenvergleiche optimieren, 3 : beinhaltet 1 und 2, 4 : <code>DEFINES</code> optimieren, 7 : beinhaltet 3 und 4,
<code>-pe</code>	Es wird in jeder Funktion der Prolog und der Epilog in den Zielcode generiert, auch wenn dazu keine Notwendigkeit besteht. Im default-Fall werden nur die Prologe und Epiloge generiert, die tatsächlich notwendig sind. Ebenso wird im default-Fall ein einfaches <code>return</code> statt des komplexen <code>RETURN-xxx</code> Makros generiert.
<code>-plain</code>	Es wird der Quellcode nach der Makroersetzung ausgegeben. Ziel der Ausgabe ist ein File mit dem Suffix <code>.i</code> .
<code>-r <filename></code>	Als Argument <code><filename></code> muß der Name eines Files angegeben werden. In diesem File werden alle mittels der <code>SOURCE</code> -Direktive von diesem Programm eingezogenen Quelltextabschnitte (auch rekursiv) dokumentiert.
<code>-scg</code>	Es erfolgt die Ausgabe der Aufrufhierarchie als HTML-File auf die Standardausgabe.
<code>-secure-logic</code>	Bei Logikausdrücken, mit deren Wert weitergerechnet wird, wird der Wert -1 bei <code>TRUE</code> erzwungen.

-setup <val>	Als Argument <val> muß der Pfadname eines Files angegeben werden, welches anstelle des Files <code>tal.stp</code> benutzt werden soll. Der Pfadname kann relativ sein. Für den Fall, daß das File <code>tal.stp</code> im aktuellen Verzeichnis steht, reicht die Angabe des Filenamens (Erläuterung zu <code>tal.stp</code> siehe Abschnitt 44).
-sql	Ausgabe von Zusatzinformationen zu SQL-Statements.
-syscall	Es werden alle Systemrufe, die im File <code>tal.stp</code> angegeben wurden, auf der Standardausgabe protokolliert, wenn sie im Programm vorkommen (Erläuterung zu <code>tal.stp</code> siehe Abschnitt 44).
-t <filename>	Der Syntaxbaum für das zu analysierende TAL-Programm wird in Form eines HTML-Files in <filename> abgespeichert.
-tg <val>	Angabe der Zielarchitektur, für die Code generiert werden soll (z.B. tandem).
-uc	Bei Angabe dieser Option werden Kommentare, welche nicht in den generierten C/C++-Code übernommen werden können, auf die Standardausgabe ausgegeben.
-uid	Verwendung einer eindeutigen Identifikationsnummer zur Kennzeichnung von herausgezogenen lokalen Variablen anstelle des Prozedurnamens.
-unused	Es werden alle ungenutzten Variablen, nicht gerufene Prozeduren und Subprozeduren auf der Standardausgabe protokolliert.
-version	Ausgabe von Versionsangaben des Translators über die Standardausgabe.
-W <val>	Es werden Warnungen der Stufe - <val> ausgegeben. <val> kann einen Wert zwischen 0 und 5 annehmen. Beim Konvertieren werden eine Reihe von Warnungen generiert, die mit dieser Option selektiv unterdrückt werden können. Je höher die Stufe, um so weniger Warnungen werden ausgegeben. <val> mit einem Wert 0 zeigt alle Warnungen an.

6.2 Die durch *cgn* generierten Files

Der *cgn* erzeugt während eines Konvertierungslaufes eine Reihe von Files, von denen allerdings nur das File mit der Endung `cp` den eigentlichen aus TAL-Quelltext konvertierten C/C++-Code enthält. Ein Headerfile mit der Endung `h` wird erzeugt, ist aber zur Zeit leer. Seine Nutzung ist für spätere Versionen vorgesehen.

Die Generierung eines Files mit Tandem-spezifischem Namen wird durch die Translatoroption `-tg tandem` erzwungen. Soll eine Portierung des generierten Files auf eine Tandem-Architektur erfolgen, ist die Nutzung dieser Option notwendig. Es können auch C/C++ Files mit den Endungen `.c` und `.h` (für UNIX) generiert werden. Das Konvertieren von TAL-Programmen nach C/C++, um diese auf UNIX bzw. Windows ausführen zu können, ist nur sinnvoll für kleine Demonstrationsprogramme. Kommerzielle Anwendungen scheitern daran, daß unter UNIX und Windows keine Implementierung der Guardian System Calls von Tandem existiert. Die Namen der generierten Files werden nach folgenden Kriterien erzeugt:

1. Wird C/C++-Code für Tandem erzeugt, so wird an den Namen des TAL-Programmes der Suffix `cp` bzw. `h` angehängt.
2. Wird C/C++-Code für UNIX erzeugt, so wird an den Namen des TAL-Programmes der Suffix `.c` bzw. `.h` angehängt.
3. Alle weiteren, generierten Files bestehen aus dem Namen des originalen TAL-Programmes, erweitert um einen Suffix, der durch einen Punkt vom Filenamem getrennt ist.

Im einzelnen entstehen folgende Files:

1. **<filename>cp**
Dieses File enthält den erzeugten C/C++-Code. Es entsteht nach einem Lauf des *cgn* aus einem vorgegebenen TAL-File, welches mit dem Präprozessor *tp* erzeugt wurde.
2. **<filename>h**
Dieses File ist für spätere Nutzungen vorgesehen, es ist zur Zeit leer und damit für eine Weiterbearbeitung des entstehenden C/C++-Codes nicht erforderlich.
3. **<filename>i**
Das File entsteht, wenn der *cgn* mit der Option `-plain` aufgerufen wird. Es enthält den TAL-Code, der nach der Auslösung aller `DEFINE`-Anweisungen entstanden ist. Hier sind als Rückbezüge zum Quellfile Kommentare folgender Form eingefügt:
 - `/*cgnline filename.tln: 5 */`

Hier ist *cgnline* ein fest vorgegebenes Wort, *filename.tln* ist der Name des Ausgangsfiles, die folgende Ziffer beschreibt die Zeilennummer, in der diese Anweisung bzw. Deklaration im Ausgangsfile anzutreffen war.
 - Eingesetzte Makros werden in folgender Form dokumentiert:

`~v:name> ersetzung ~v:name<`

Hier ist *v* eine natürliche Zahl, die beginnend mit 1 die Tiefe der Ersetzungen angibt. Wird also ein Makro in einem anderen benutzt, so ist dessen Tiefe um eines höher als die des nutzenden Makros. *ersetzung* ist der substituierte TAL-Quelltext, *name* der Name des genutzten Makros. Dieses File dient rein informativen Zwecken, es wird nicht weiterverarbeitet.
4. **<filename>sql**
Dieses File beinhaltet Informationen zu den im TAL-Quelltext benutzten SQL-Anweisungen. Es wird angelegt, wenn der *cgn* mit der Option `-sql` gestartet wird.

6.3 Die Gestaltung der Meldungen

Der *cgn* erzeugt während eines Translatorlaufes eine Reihe von Meldungen, welche folgenden beispielhaften Aufbau besitzen:

```
demo11.tln Warnung G152 Zeile 541: Funktion BERECHNEN ohne RETURN  
(eingefuegt)  
translation sp1v2q.tln: 0 Fehler, 1 Warnungen und 0 Sorrys
```

Jede Ausschrift dokumentiert entweder einen Fehler, eine Warnung oder ein Sorry.

Fehler

sind Eigenschaften des TAL-Programmes, die eine korrekte Konvertierung nicht erlauben. Treten Fehler während eines Translationslaufes auf, ist das TAL-Programm syntaktisch oder semantisch inkorrekt. Eine weitere Bearbeitung ist nicht möglich.

Warnungen

zeigen Eigenschaften des TAL-Programmes an, die eine Konvertierung zulassen - der entstandene C/C++-Code ist compilierbar und ausführbar. Bei bestimmten Warnungen ist zu prüfen, ob der generierte C/C++-Code semantisch korrekt arbeitet. Werden diese Warnungen ignoriert, können Laufzeitfehler auftreten. Diese Gruppe von Warnungen ist in der Nutzerdokumentation im Anhang B mit einem ! (Ausrufezeichen) in der Spalte ART gekennzeichnet.

Sorrys

dokumentieren Eigenschaften des Translators, die eine Konvertierung des TAL-Programmes nicht zulassen, da bestimmte TAL-Sprachkonstruktionen prinzipiell nicht realisiert werden können bzw. in der gegenwärtigen Version des TTC noch nicht realisiert wurden.

Im weiteren wird allgemein von Meldungen gesprochen, wenn sowohl Fehler, Warnungen als auch Sorrys gemeint sein können.

Jede Ausschrift im Protokoll wird eingeleitet mit dem Namen des Files, in dem die Meldung auftritt. Diese Lokalisierung ist für Programme, die nicht mit *tal_only* behandelt wurden, so, daß der Name des Files dem Originalfilenamen entspricht, welches durch eine `?SOURCE`-Direktive eingezogen wurde. Ist das File mit *tal_only* behandelt worden, steht als Filename ausschließlich der zur Verfügung, der dem Translator übergeben wurde.

Desweiteren wird die Art der Meldung und eine interne vierstellige Fehlernummer angezeigt. Die korrekte Positionierung des Fehlers wird durch die angegebene Zeile möglich. Die Zeilennummern beziehen sich standardmäßig auf die Lokalisierung im Originalfile. Bei Angabe der `-im`-Option erfolgt die Generierung der Zeilennummer bezogen auf das File `<filename>.tln` bzw. `<filename>.tln~`.

Jede Meldung ist durch einen Kommentar versehen, der die Bedeutung der Meldung erklärt. Alle Meldungen des *cgn* sind im Anhang B dokumentiert.

6.4 Die generierten Zerteilerinformationen

Die Zerteilung einzelner Komponenten des C-Files ist steuerbar. Im File zur Option `-dir` besteht die Möglichkeit, einzelne Files bzw. Sektionen anzugeben, die nicht weiter zerteilt werden sollen. Dabei bestehen folgende Möglichkeiten:

1. name
2. +FILE
3. +FILE:SECTION
4. -FILE
5. -FILE:SECTION
6. *FILE
7. *FILE:SECTION

Die Angabe der Objekte in diesem File geschieht zeilenweise, wobei jede Zeile nur einen Objektnamen enthalten darf, welcher der TAL-Name (nicht der erzeugte C-Name) des Objektes sein muß. Ist name angegeben, so wird für diese globale Übersetzungseinheit (Prozedur, Variable, ...) keine Zerteilerinformation erzeugt. Komponenten, die aus FILE bzw. FILE:SECTION stammen, werden nicht mit Zerteilungsinformationen generiert. Steht ein + davor, so wird für FILE bzw. FILE:SECTION und alle dort eingezogenen Files keine Zerteilerinformation erzeugt, steht ein - davor, so wird FILE bzw. FILE:SECTION zerteilt, nicht aber die dort eingezogenen Quellen. Steht ein *, so werden Zerteilerinformationen für alle in FILE/FILE:SECTION eingezogenen Files, nicht aber für FILE/FILE:SECTION selbst unterdrückt. FILE bezieht sich auf ein komplettes File. Diese Direktive ist nur für `?SOURCE FILE`-Direktiven wirksam. FILE:SECTION bezieht sich auf Sections in einem File. Diese Direktive ist nur für `?SOURCE FILE(SECTION)` Direktiven wirksam. Die Angabe FILE kann für alle drei Versionen optional mit einem FILE:* erweitert werden. In diesem Fall bezieht sich die Information auf alle Sektionen eines Files.

6.5 Die Behandlung der `INVOKE`-Strukturdefinitionen

Strukturdefinitionen, die in TAL-Quellen durch den `INVOKE`-Mechanismus bekannt gemacht werden, sind dem *cgn* unbekannt. Aus diesem Grund ist eine zusätzliche Bearbeitungsstufe des TAL-Programmes notwendig.

6.5.1 Aufbau der Eingabedatei

Eingabe für die gesamte Analyse ist ein File, welches durch den Anwender aus der Umgebung der Datenbank erstellt werden muß. Dieses enthält alle Strukturdefinitionen, die im zu konvertierenden TAL-Programm als `INVOKE`-Strukturen benutzt werden sollen. Die Definition der Strukturen muß in der Sprache TAL erfolgen. Ändern sich Datenbankstrukturen, so muß dieses File den neuen Gegebenheiten angepaßt werden. Eine feste Vorgabe dieses Files kann damit nicht erfolgen. Für das Format gilt:

1. Groß- und Kleinschreibung ist erlaubt, Kommentare nach TAL-Syntax dürfen enthalten sein. Das File sollte also einen solchen Aufbau besitzen, daß es vom Präprozessor bearbeitet werden kann.
2. Das File enthält TAL-Strukturtemplate-Definitionen, die aus einer `INVOKE`-Anweisung hervorgegangen sind. Die Syntax dieser Strukturbeschreibung muß TAL-Syntax sein.
3. Jede Strukturdefinition muß im Format dem folgenden Beispiel gleichen:

```
SECTION K^QPPK AS K^QPPK^DEF;
! Record Definition for table ``W01.$DATA1.DATAW.DS0CPD
! Definition current at 11:33:05 - 03/17/97
STRUCT K"QPPK"DEF(*);
    string pr^plan[0:7] /DECIMAL(8) UNSIGNED;/;
    string pr^plan^index[0:1] /DECIMAL(2) UNSIGNED;/;
    string sachnr[0:11];
END;
```

Hier sind folgende Aspekte zu beachten:

- (a) Vor `SECTION` darf kein Fragezeichen stehen, damit der Präprozessor hier keine `SECTION`-Verarbeitung anstößt.
- (b) Nach `SECTION` folgt der Name der SQL-Tabelle.
- (c) Optional darf eine Sequenz `AS <talname>` folgen. Hier ist `<talname>` der Name des Strukturtemplates in TAL. Fehlt die `AS`-Angabe, so wird entsprechend der Manuals der Name des Strukturtemplates aus dem Namen der SQL-Tabelle mit einem angehängten `^TYPE` gebildet.
- (d) Es dürfen beliebig viele Kommentarzeilen folgen.
- (e) Es folgt die Strukturtemplatedefinition. Der Name der Struktur muß identisch mit dem im `SECTION`-Bereich angegebenen sein, andernfalls erfolgt eine Warnung.
- (f) Die Struktur wird in TAL-Syntax beschrieben und muß demzufolge mit einem terminierenden `END` versehen sein.
- (g) Die Schlüsselworte `SECTION` und `STRUCT` müssen die ersten signifikanten Worte auf der jeweiligen Zeile sein, Leerzeichen und Tabulatoren dürfen davorstehen, sonst nichts.

Es erfolgt eine vollständige Umsetzung aller SQL-Anweisungen, die mit dem `EXEC`-Befehl beschrieben sind.

Es existiert folgende Ausnahme:

- Option `-C` NICHT gesetzt:

(Es soll C++-Code generiert werden.)

Aufgrund der in der gegenwärtigen Version des C++-Compilers enthaltenen Einschränkungen (NonStop SQL wird nicht unterstützt.) werden nach der Konvertierung alle eventuell auftretenden `EXEC`-Anweisungen in Kommentare eingeschlossen, so daß eine Compilierung dieser C++-Quellen mit dem C++-Compiler möglich ist. Allerdings sind die so entstandenen Objekte nicht lauffähig, da die entsprechenden `EXEC`-Anweisungen ausgeschlossen wurden.

6.5.2 Generierung von Dateien für die INVOKE-Behandlung

Während des Transformationsprozesses für die INVOKE-Behandlung werden 2 Dateien erzeugt, welche Indexinformationen bzw. die Strukturdefinitionen der INVOKE-Strukturen enthalten. Für die Transformation sind folgende Schritte erforderlich:

- Aus der Umgebung der Datenbank ist eine Datei zu erzeugen, welche alle Strukturdefinitionen enthält, die im zu konvertierenden TAL-Programm als INVOKE-Strukturen benutzt werden sollen. Der Aufbau dieser Datei ist im Abschnitt 6.5.1 beschrieben.
- Die so erstellte Datei wird mit dem Präprozessor *tpp* und die resultierende *.tln*-Datei anschließend mit dem Programm *tal_only* bearbeitet.
- Die auf diese Weise erzeugte Ergebnisdatei muß mit folgendem Kommando unter *cygwin* bearbeitet werden:

```
awk -f ivkdefs.awk -v ccode=<idxfile> -v cstrct=<sctfile>
[sqlstruct.ivk] < <ivkfile>
```

Zu diesem Kommando gelten die folgenden Festlegungen:

- Unter der gegebenen *cygwin*-Installation muß mindestens eines der Werkzeuge *awk*, *nawk* oder *gawk* vorhanden sein.
- *ivkdefs.awk* ist ein im Umfang des Translators enthaltenes Skript.
- *idxfile* ist eine Datei, die während dieses Transformationsprozesses geschrieben wird. Sie enthält Indexinformationen, die im weiteren noch beschrieben werden.
- *sctfile* ist eine Datei, die während dieses Transformationsprozesses geschrieben wird. Sie enthält alle Strukturdefinitionen.
- *ivkfile* ist jene Datei, die aus den Strukturbeschreibungen nach der Bearbeitung mit *tpp* und *tal_only* entstanden ist.
- *sqlstruct.ivk* ist eine Datei, die Beschreibungen der Basisstrukturen wie SQLCA oder SQLSA enthält. Diese Datei ist ebenfalls im Lieferumfang des Translators enthalten. Enthält das zu übersetzende TAL-Programm SQL-Direktiven der Form

```
EXEC SQL INCLUDE SQLCA;
```

so ist *sqlstruct.ivk* in jedem Fall mit anzugeben, werden diese Strukturen im TAL- Programm nicht genutzt, kann die Angabe entfallen.

Die erzeugte Indexdatei hat folgenden Aufbau:

- Jeder Indexeintrag steht in einer Zeile.
- Jeder Indexeintrag besitzt 4 Komponenten:
 1. einen Index.
Dieser setzt sich aus dem Namen der SQL-Tabelle und dem Namen der zugehörigen TAL-Struktur zusammen. Beide sind durch ein # getrennt. Dieser Index wird intern als Index in einer Tabelle benutzt, um jedes auftretende INVOKE eindeutig einer Tabellendefinition zuordnen zu können.
 2. den Namen der SQL-Tabelle,
 3. den Namen der TAL-Struktur,
 4. die Zeile im Strukturfile, in welcher die Strukturdefinition beginnt.

6.6 Linken von verschiedenen Modulen

6.6.1 Initialisierung globaler Daten

Ausführbare Programme können prinzipiell aus verschiedenen Modulen bestehen. Jeder Modul entsteht dabei durch einen separaten Konvertierungsprozeß aus einem TAL-File. Eines dieser TAL-Files muß eine TAL-Funktion mit dem Attribut `MAIN` enthalten. Im weiteren gelten folgende Festlegungen:

1. Die Ausführungen beziehen sich grundsätzlich auf den aus TAL generierten C/C++-Code.
2. Die Funktion, welche in C/C++ aus der TAL-Funktion mit dem Attribut `MAIN` entstanden ist, wird `MAIN`-Funktion genannt.
3. Das Modul, welches die `MAIN`-Funktion beinhaltet, soll als Topmodul bezeichnet werden.
4. Modulen, welche keine `MAIN`-Funktion enthalten, werden als Submodul bezeichnet.

Für Programmpakete, die durch den Translator `cg` erzeugt wurden, existiert ein Macro mit dem Namen `MAIN_FUNCTION__`. Dieses Makro beinhaltet den Ruf der `MAIN`-Funktion sowie den Ruf verschiedener Initialisierungsfunktionen für das Laufzeitsystem `tcsys`.

Zusätzlich wird von `cg` der Ruf einer Funktion `_Ar_init` generiert. Diese befindet sich in dem File, in dem auch die `MAIN`-Funktion implementiert ist. `_Ar_init` selbst initialisiert globale Nutzerdaten, die in diesem File vereinbart sind.

Für jedes File, welches keine `MAIN`-Funktion enthält, werden diese Initialisierungen durch Funktionen vorgenommen, deren Namensbildung nach dem Schema:

```
_Ar_init_xxx
```

erfolgen, wobei `xxx` der Name des gegebenen Files ist.

Diese Funktionen werden nicht automatisch gerufen. Sie müssen aber in jedem Fall abgearbeitet werden, um Daten von Submodulen korrekt vor Programmbeginn zu initialisieren.

Zur Unterstützung dieses Prozesses generiert der Translator in jede Implementierung der Funktion `_Ar_init` den Aufruf einer Funktion `_Ar_init_submodule`. Besteht ein Paket definitiv nur aus einem, eine `MAIN`-Funktion enthaltenden Modul, so kann der Translator mit der Option `-nolink` gestartet werden. In diesem Fall wird der Ruf von `_Ar_init_submodule` unterdrückt. Die genannte Funktion kann auf zwei verschiedenen Wegen erzeugt werden:

1. Sie wird "per Hand" geschrieben. Dann muß sie folgendes Aussehen haben:

```
void
_Ar_init_submodule()
{
    extern void _Ar_init_xx1();
    extern void _Ar_init_xx2();
    ...
    extern void _Ar_init_xxn();
    _Ar_init_xx1();
    _Ar_init_xx2();
    ...
    _Ar_init_xxn();
}
```

Im Anweisungsteil der Funktion werden sequentiell die Initialisierungsfunktionen aller Submodulen gerufen. Die so entstehende Funktion muß gemeinsam mit den anderen zum Paket gehörenden Modulen gelinkt werden.

2. Es wird das zum Lieferumfang des Translators gehörende Programm *itc* benutzt, welches im weiteren beschrieben wird. Hier wird die Erzeugung der oben genannten Funktion `_Ar_init_submodule` automatisiert.

6.6.2 Das Werkzeug *itc*

itc ist ein Werkzeug zur Erzeugung der Initialisierungsfunktion `_Ar_init_submodule`. Der Aufruf von *itc* ist

```
itc <option 1> ... <option n> <file1> <file2> ...
```

Dabei sind `<file1>` `<file2>` ... alle zum linken Paket gehörenden C-Files, die durch den Translator *cgn* erzeugt wurden. Für *itc* existieren die folgenden Optionen:

itc-Optionen:

Option	Bedeutung
<code>-o <filename></code>	Die zu generierende Funktion <code>_Ar_init_submodule</code> wird in das File <code><filename></code> geschrieben. Ist diese Option nicht angegeben, so wird das Ergebnis des <i>itc</i> -Laufes auf die Standardausgabe geschrieben.
<code>-h</code>	Anzeige einer Kurzhilfe.

Der Translator erzeugt für alle Submoduln, die eine Funktion `_Ar_init_xxx` besitzen, eine Warnung G588 (Vgl. Abschnitt B).

Zur Steuerung des Linkens existiert eine Translatooption `-nolink` (Vgl. Abschnitt 23).

6.7 Das Konvertieren von TAL-DEFINE's

DEFINES in TAL können nicht in jedem Fall automatisch von *cgn* in C/C++ Makros umgesetzt werden, da keine Garantie dafür besteht, daß es sich um abgeschlossene Sprachkonstrukte handelt. Daher ersetzt der Translator vor der Analyse den DEFINE-Aufruf durch den entsprechenden DEFINE-Körper. Der so entstandene TAL-Code wird konvertiert.

Es gibt jedoch zwei Ausnahmen von dieser Regel, die optional verwendet werden können.

1. Wenn ein DEFINE-Körper genau eine TAL-Konstante ist, so ist die Umsetzung in ein oder mehrere C/C++ Makros möglich. Dazu ist der Wert `<val>` der Option `-o` auf 4 zu setzen (siehe Beschreibung dieser Option). Drei Fälle sind dabei zu unterscheiden:
 - Ist der DEFINE-Körper eine Zahlkonstante (egal welchen Typs), so wird genau ein C/C++ Makro erzeugt, dessen Makrokörper wieder diese Konstante im C/C++ Format ist.

Beispiel:

```
DEFINE A = 1234#;           # define a 1234
```

- Ist der DEFINE-Körper in TAL eine STRING-Character-Konstante (charakterisiert durch doppelte Hochkommas "xyz") mit einer Länge `<= 4` Zeichen, so werden drei C/C++ Makros zur unterschiedlichen Verwendung erzeugt. Zur Unterscheidung dieser drei Varianten wird eine Ziffer angehängt. Anhand dieser Suffixe wird bei der Verwendung solcher Makros über die korrekte Auswahl entschieden.

Beispiel:

```
DEFINE A = ''1234''#;      # define A_4 '1234'
                          # define A_2 {'1','2','3','4'}
                          # define A_1 ''1234''
```

Dabei dient:

```
# define A_1 ''1234''
als eine C-Stringkonstante für den Eintrag in die Initialisierungstabellen,

# define A_4 '1234'
für die Verwendung als Zahl (nur bei einer Länge <= 4 Zeichen und bei Ziffernzeichen),

# define A_2 {'1','2','3','4'}
für eine C-Initialisierungsliste.
```

- Ist der DEFINE-Körper eine STRING-Character-Konstante mit einer Länge > 4 Zeichen, ist eine Konvertierung als long-Konstante (also in einfachen Hochkommas) nicht möglich und es werden nur die Initialisierungsliste (A_2 im Beispiel) und C-Zeichenkettenkonstante (A_1) erzeugt.

Beispiel:

```
DEFINE A = 'Sommer' '#;                                # define A_2
                                                         {'S','o','m','m','e','r'}
                                                         # define A_1 'Sommer''
```

Steht das DEFINE in TAL innerhalb einer Prozedur oder Subprozedur, so ist es auch nur dort gültig. In C/C++ werden unmittelbar nach dem Ende des Funktionskörpers entsprechende #undef- Anweisungen generiert. Namen von #define's in C/C++ werden generell mit Großbuchstaben erzeugt.

2. Handelt es sich beim DEFINE-Körper um eine vollständige Anweisung oder einen kompletten TAL- Ausdruck, so ist es möglich, den Aufruf des Makros als C/C++ Funktions- bzw. Makroaufruf zu generieren. Die Umsetzung des Makros als #define selber muß jedoch von Hand erfolgen.

Folgende Voraussetzungen müssen erfüllt sein, damit eine solche Umsetzung erfolgt:

- Der folgende Eintrag muß im File tal.stp im letzten Abschnitt (nach ~END_PROTOTYPES) stehen:

```
~@ Spezialmakro fuer interne Zwecke -- bitte nicht entfernen
~PROC $MAKRO
    [$MAKRO, $MAKRO_GG, __MAKRO__, ~SIMPLE_STDFUNC, (GG) ]
    (~GENERIC, ~GENERIC)
    ~RETURNS (INT);
```

Auf die Bedeutung dieses Eintrags kann hier nicht eingegangen werden. Er ist nur für die interne Verwaltung der Makros wichtig. Da er allein wirkungslos ist, sollte er generell in jedem File tal.stp zu finden sein.

Das eigentliche Makro ist wie eine Standardfunktion in das File tal.stp aufzunehmen. Im Kapitel 9 dieser Nutzerdokumentation ist der Aufbau eines solchen Eintrags beschrieben. Die Typen der Parameter sind entsprechend den Erfordernissen zu setzen. Meist können DEFINE-Parameter jedoch unterschiedliche Typen haben (Bei der ersten Verwendung wird z.B. als zweiter Parameter ein INT-Ausdruck übergeben und bei einer weiteren eine Struktur.). Daher ist es bei DEFINES sinnvoll, alle Parameter vom Typ GENERIC anzugeben. Der Name, den das Makro in C/C++ erhalten soll, kann beliebig entsprechend der C/C++ Namenskonventionen gewählt werden.

Beim Rückgabetyt ist es ein Unterschied, ob der Körper des zu übersetzenden Makros eine Anweisung oder einen Ausdruck darstellt. Dabei sind die folgenden Fälle zu unterscheiden:

- Der Körper ist eine Anweisung:

In diesem Fall ist als Rückgabetyt generell VOID zu verwenden.

Beispiel:

```
DEFINE SET^SIZE(A,B) = A := $LEN(B)*$OCCURS(B)#;
```

benötigt den folgenden Eintrag im File tal.stp:

```
~PROC SET^SIZE_GG  
[SET^SIZE_GG, SET^SIZE_GG_GG, set_size, ~SIMPLE_STDFUNC,  
(GG)]  
(~GENERIC, ~GENERIC)  
~RETURNS (~VOID);
```

In diesem Fall wird der Aufruf

```
SET^SIZE(X.SIZE,X)
```

als

```
set_size(x.size,x)
```

übersetzt. Das Makro set_size() könnte z.B. wie folgt selber definiert werden:

```
#define set_size(a,b) a = sizeof(b)
```

- o Der Körper ist ein Ausdruck:

Dabei muß als Rückgabetyt der Typ, den der vom Ausdruck berechnete Wert in TAL hat, angegeben werden.

Beispiel:

```
DEFINE ^LEN(F) = ($LEN(F)*$OCCURS(F))#;
```

benötigt den folgenden Eintrag im File tal.stp:

```
~PROC ^LEN_G  
[^LEN, ^LEN_G, sizeof, ~SIMPLE_STDFUNC, (G)]  
(~GENERIC)  
~RETURNS (INT);
```

In diesem Fall wird

```
S := ^LEN(X.Y)
```

als

```
s = sizeof(x.y)
```

umgesetzt. Hier kann auch die Definition von Hand entfallen, da sizeof() ein C-Operator ist.

Sind die beiden Fälle erfüllt, so erfolgt die Umsetzung von Makros in der eben beschriebenen Weise, ohne daß zusätzlich die Angabe einer Option erforderlich wäre.

Durch die beiden vorgestellten Methoden wird ein großer Teil der in TAL verwendeten Makros abgedeckt. Im Ergebnis der Konvertierung entsteht übersichtlicherer C/C++ Code als der bei reiner Makroexpansion.

6.8 Das Initialisieren von TAL-Strukturen

In TAL existiert keine Möglichkeit, Strukturen direkt zu initialisieren. Strukturinitialisierungen werden programmtechnisch so realisiert, daß ein initialisiertes TAL-Feld eine TAL-Struktur überlagert:

Beispiel:

Gegeben sei folgendes Programmfragment in TAL:

```
INT FELD[0:3] := [ 1,2,3,4 ];
STRUCT X = FELD;
BEGIN
  INT A,B,C,D;
END;
```

Diese Deklarationen befinden sich auf globaler Programmebene. Durch die Überlagerung von "X" mit "FELD" werden die einzelnen Komponenten der Struktur "X" initialisiert. Eine erste Konvertierungslösung des TTC nach C/C++ erzeugt ebenfalls ein initialisiertes Feld, welches die zu initialisierende Struktur überlagert. Das obige Programmfragment wird mit dieser ersten Lösung wie folgt konvertiert:

```
short feld[4]= {1,2,3,4};
struct Tx {
    short a, b, c, d;
} *x= (struct Tx*)feld;
```

Das entspricht der Forderung nach semantischer Äquivalenz zwischen TAL- und C/C++ Programmen. Es werden für alle Anwendungsprogramme korrekte Initialisierungen generiert. Insbesondere bei TAL-Programmen, bei denen komplexe Strukturen zu initialisieren sind, erschwert die obige indirekte Initialisierung durch Überlagerung mit einem Feld die Wartbarkeit des resultierenden C/C++ Codes. Hinzu kommt, daß es in C/C++ durchaus möglich ist, Strukturen direkt zu initialisieren. Auf dieser Erkenntnis basiert die zweite mögliche Konvertierungsstrategie. Die indirekte Initialisierung durch Überlagerung einer TAL-Struktur mit einem initialisierten Feld wird im resultierenden C/C++ Code durch eine direkte Strukturinitialisierung ersetzt. Für das obige TAL-Beispiel wird generiert:

```
/* array FELD ignored (direct struct initialization) */
struct Tx {
    short a, b, c, d;
} x= -1,2,3,4•;
```

Das zur Überlagerung verwendete TAL-Feld wird nicht mit in den C/C++ Code generiert. Stattdessen erfolgt die Ausgabe eines Kommentars, daß an dieser Stelle im originalen TAL-Programm ein Feld deklariert war. Das Feld selbst wird deshalb nicht generiert, weil es bei einer direkten Initialisierung überflüssig ist. Folgende Restriktionen existieren:

- Die Methodik bezieht sich nur auf Felder/Strukturen auf globaler Ebene.
- Die TAL-Struktur kann dimensioniert sein. TTC berechnet für jede Strukturkomponente deren Größe und erzeugt aus der Initialisierungsliste des TAL-Feldes eine Initialisierung derart, daß jeder Strukturkomponente eine ihrer Größe entsprechende Konstante exakt zugeordnet ist. Die Initialisierung ist dann möglich, wenn sowohl Struktur als auch Feld aus INT/STRING-Komponenten bestehen.
- Die Struktur darf keine Redefines enthalten. Diese Einschränkung kann durch den Konfigurationsparameter DIRECTINITIAL REDEFINE aufgehoben werden. In diesem Fall muß der Anwender selbst prüfen, ob die Initialisierung korrekt ist.
- Das Feld muß initialisiert sein.
- Die Struktur kann dimensioniert sein.
- Befinden sich im Feld mehr Initialisierungselemente als die Struktur verkraften kann, dann gehen diese Elemente verloren.
- Befindet sich das Feld in einem Header und dient zur Initialisierung mehrerer Strukturen, so kann auch nach einer evtl. Zerteilung dieses eine Feld nicht mehr hergestellt werden. Das besagt, daß Änderungen, die bislang im Feld erfolgten, parallel an allen (!)

Strukturen, die jetzt direkt durch die Initialwerte des Feldes belegt werden, durchgeführt werden müssen.

- Die Strukturvariable wird bei einer direkten Initialisierung nicht mehr als Zeigervariable, sondern als direkte Strukturinstanz erzeugt.
- Befinden sich Feld und/oder Struktur innerhalb einer `BLOCK`-Deklaration und wird die `-eb`-Option gesetzt, so entfällt die Initialisierung der Struktur, da diese extern angelegt wird. In diesem Fall muß der Anwender dafür sorgen, daß es genau einen Modul gibt, in dem die Initialisierung der Struktur erfolgt. Diese Struktur muß in allen Modulen, die zum Paket gehören, als `DIRECTINITIAL` generiert werden. Andernfalls kann es zu zwei verschiedenen Definitionen kommen (einmal als Zeiger, einmal als direkte Strukturinstanz).

Die beschriebene Lösung besitzt Auswirkungen auf die Fälle, in denen das initialisierte TAL-Feld nicht nur zur Strukturinitialisierung, sondern auch in anderen Zusammenhängen im Programm genutzt wird. In diesen Fällen müssen Zugriffe auf dieses Feld innerhalb des Programmes manuell umgeschrieben werden. Gesteuert wird die Strukturinitialisierung über Konfigurationsparameter der Datei `tal.cfg`:

- `DIRECTINITIAL: automatic/n`:
schaltet den Initialisierungsmodus. Bei "n" wird die indirekte Initialisierung der Struktur durch Überlagerung mit einem Feld generiert. Bei "automatic" wird durch TTC über die bestmögliche Initialisierung entschieden.
- `DIRECTINITIAL_STRING: y/n`:
Wenn "y", so wird in der direkten Stringinitialisierung die Erzeugung von Zeichenketten statt Zeichenfeldern erzeugt. Anstelle 'x','y' wird "xy" erzeugt. Das ist nur dann möglich, wenn Code für einen C-Compiler erzeugt wird. C++ läßt so etwas nicht zu.
- `DIRECTINITIAL_REDEFINE: y/n`:
Wenn "y", so werden Strukturen auch dann direkt initialisiert, wenn Redefine-Komponenten enthalten sind (default-Wert: Keine Initialisierung).

7 Das File ttc.cfg

Dieser Abschnitt beschreibt den Aufbau des Files `ttc.cfg`. Mit Hilfe dieses Files kann der Nutzer das Layout des zu generierenden C/C++-Codes beschreiben. Mittels der Option `-CFG <filename>` kann statt des Files `ttc.cfg` ein File mit dem Namen `<filename>` zur Layoutbeschreibung angegeben werden. Fehlt die Option `-CFG`, dann verwendet `cgn` das File `ttc.cfg` im aktuellen Verzeichnis zur Formatierung. Fehlt dieses, treten Standardannahmen zur Formatierung in Kraft. Die Standardannahmen entsprechen den default-Werten der nachfolgenden Tabelle.

Das Formatierungsfile besitzt folgenden Aufbau:

1. Jeder Parameter mit seinem Wert muß in einer eigenen Zeile stehen.
2. Jede Zeile muß mit einem Newline abgeschlossen sein (auch die letzte).
3. Zwischen Parameter und zugehörigem Wert muß ein Doppelpunkt stehen.
4. Nach dem Wert muß direkt das Newline stehen, Leerzeichen oder Tabulatoren sind nicht erlaubt.
5. Wird ein Parameter mehrfach in einem File definiert, so wird prinzipiell der erste Parameter dieses Namens gewertet.
6. Ist ein Parameter nicht angegeben, so tritt dessen default-Wert in Kraft. Das gleiche gilt, wenn der Parameter zwar angegeben ist, aber einen ungültigen Wert besitzt.

Es existieren die folgenden Parameter mit ihren Wertebereichen:

```
;*****
;
;KOMMENTARBEHANDLUNG
;
;*****
;CMTFILE: leer/!name?
; !name?: Name des Kommentarfiles
; default:!programmname?.cmt
;*****
;INSERT_COMMENT: y/n
; Wenn y, dann
; Ruecktransformation der Kommentare in den generierten Code.
; default: n
;*****
;PRINT_UNREAD_COMMENTS: y/n
; Falls y, so werden am Ende der Uebersetzung
; alle Kommentare ausgegeben, die nicht ruecktrans-
; formiert wurden.
; (gleichbedeutend zu Option -uc)
; default: n
;*****
;TPP_COMMENTS: y/n
; Falls nicht auf n gesetzt, werden alle
; Preprozessordirektiven als Kommentare mit in das
; Zielfile kopiert.
; default: y
;*****
;EMPTY_COMMENTS: y/n
; Falls nicht auf n gesetzt, werden auch leere
; Kommentare umgesetzt.
; default: y
;*****
;WHITE_COMMENTS: y/n/skipline
; Falls nicht auf n gesetzt, werden auch Kommentare
; umgesetzt, die nur aus whitespaces bestehen.
; Falls skipline gesetzt,
; so werden leere Kommentar als Leerzeile eingefuegt,
```

```

; falls diese allein auf einer Zeile in TAL gestanden
; haben.
;*****
;PROC_PRECOMMENTS: y/n
; Wenn auf n gesetzt, werden die Prozedurvorkommentare
; (/ * TAL-Prozedur MMM (Zeile 9) -? C-Funktion Mmm * /)
; nicht generiert.
; default: y
;*****
;CPP_COMMENTS: Falls auf y gesetzt, werden Kommentare, die
; keinen rechtsseitigen Begrenzer haben, als C++-
; Kommentare generiert (//)
; default: n
;*****
;CMT_HEURISTIC: hr_2/-
; Angabe der Heuristik, nach der Kommentare
; eingefuegt werden sollen, z.Z. nur
; 'hr_2' -? nach comment_heuristic_2 implementiert,
; hier sind
; TAL_Expr und TAL_Constant keine Stoppunkte.
;*****
;ENDCOMMENT_THEN:
;ENDCOMMENT_ELSE:
;ENDCOMMENT_ELSEIF:
;ENDCOMMENT_ELSIF:
;ENDCOMMENT_PROC:
;ENDCOMMENT_WHILE:
;ENDCOMMENT_SWITCH:
;ENDCOMMENT_FOR:
;ENDCOMMENT: y/n
; Wenn auf 'n' gesetzt, werden die
; abschliessenden Kommentare der im Suffix
; beschriebenen Bloecke ignoriert,
; wie z.B. /* end case * /
; Allgemein kann das auch durch
; Setzen von ENDCOMMENT auf 'n' erreicht werden.
; Diese Einstellung gilt dann fuer alle
; Kommentarabschluesse.
; default: y
;*****
;
;TEXTSTRUKTURIERUNG IN PARAMETERLISTEN
;
;*****
;BREAK_PARM: y/n
; Zeilenumbruch nach jedem Paraemeter in
; formalen Parameterlisten.
; default: n
;*****
;BREAK_CALL: y/n
; Zeilenumbruch nach jedem Parameter in
; Funktionsrufen.
; default: n
; Die Angabe von BREAK_CALL setzt implizit
; auch MAKRO_BREAK, GUARDIAN_BREAK und
; USER_FUNCTION_BREAK auf BREAK_CALL_LEN.
;*****
;BREAK_PARM_PROTO: y/n
; Zeilenumbruch nach jedem Paraemeter in
; aktuellen Parameterlisten von Funktionsprototypen
; (extern)
; default: n
;*****
;BREAK_CALL_LEN: !zahl?
; Nur aktiv, wenn BREAK_CALL auf y gesetzt. In diesem
; Fall wird eine Zahl erwartet, die angibt,
; nach wievielen Parametern umgebrochen wird.
; Im Fehlerfall (zahl nicht korrekt) wird
; 1 angenommen.
; default: 1
;*****

```

```

;MAKRO_BREAK: !zahl?
; Umbruch der ruecktransformierten Makros,
; nur wirksam, wenn BREAK_CALL(_LEN) nicht gesetzt ist.
;*****
;TTC_BREAK: !zahl?
; Umbruch in proetcon definierten Makros
; und Funktionen nach !zahl? Parametern,
; nur wirksam, wenn BREAK_CALL(_LEN) nicht gesetzt ist.
;*****
;GUARDIAN_BREAK: !zahl?
; Umbruch in Guardian-Calls nach !zahl? Parametern,
; nur wirksam, wenn BREAK_CALL(_LEN) nicht gesetzt ist.
;*****
;STANDARD_BREAK: !zahl?
; Umbruch in Stansardfunktionen nach !zahl? Parametern
; nur wirksam, wenn BREAK_CALL(_LEN) nicht gesetzt ist
;*****
;USER_FUNCTION_BREAK: !zahl?
; Umbruch in Nutzer-Calls nach !zahl? Parametern,
; nur wirksam, wenn BREAK_CALL(_LEN) nicht gesetzt ist.
;*****
;
;TEXTSTRUKTURIERUNG BEI ANWEISUNGEN UND DEKLARATIONEN
;
;*****
;BLOCK_STATEMENTS: y/n
; Wenn auf n gesetzt, wird versucht, die oeffnenden
; und schliessenden Klammern um Bloecke wegzulassen,
; falls moeglich.
; default: n
;*****
;CASE_BLOCK: y/n
; Wenn auf n gesetzt, werden in case-Zweigen die
; Klammern weggelassen. Das kann zu Problemen
; fuehren, wenn im C++ Modus hier Variablen
; initialisiert werden.
; default: y
;*****
;DCLR_MODE: break/nobreak
; Angabe, ob in Deklarationen umgebrochen werden soll
; oder nicht.
; default: nobreak
;*****
;DCLR_LEN: count,!zahl?/init,!zahl?
; Angabe, wie Umbrueche in Deklarationen
; erfolgen sollen. Voraussetzung ist das Setzen von
; DCLR_MODE auf break.
; count,!zahl?: Umbruch nach !zahl? Definitionen
; default: count,1
; init,!zahl?: wie count,!zahl?, nur zwangsweiser
; Umbruch nach jeder initialisierten Variablen
; default: count,1
;*****
;DCLR_INSERT: fix/variable
; Bestimmt die Anzahl der Einrueckungen auf der
; Folgezeile bei DCLR_MODE: break.
; fix :Einrueckung generell um die aktuelle
; Einrueckungserhoehung
; variable: Folgezeilen beginnen generell unter
; der Variablendefinition, die in der ersten
; Zeile dieser Definition am Anfang steht.
; default: fix
;*****
;
;BEZEICHNERNAMEN
;
;*****
;PROCVARNAME: extend/noextend
; Bei noextend wird bei der Erzeugung von globalen Namen
; (Name_GlobalID_Proc,Name_GlobalId_UID,

```

```

; Name_ParamAssign,Name_Shadow_Parm,
; Name_GlobalType_UID,Name_GlobalType_Proc,
; Name_GlobalType)
; auf eine Erweiterung durch den
; Prozedurnamen verzichtet, wenn trotzdem
; Eindeutigkeit vorliegt (heuristisch).
; default: extend
;*****
;SUBPROCNAME: extend/noextend
; Subprozedurnamen werden nicht erweitert, wenn
; noextend gesetzt ist und Namenseindeutigkeit
; vorliegt.
; default: extend
;*****
;CIRCUMFLEX: Gibt an, durch welche Zeichenfolge das ` zu
; ersetzen ist. Hier kann eine beliebige Zeichenfolge
; stehen, der Translator wirft alle Zeichen, die
; nicht fuer C-Bezeichner zulaessig sind, raus.
; default: '`_`'
;*****
;UNDERSCORE: extend/noextend
; Gibt an, ob Unterstrich erweitert wird oder nicht.
; default: extend
;*****
;PROCNAMESTART: upper/lower
; Bei Angabe upper wird der erste Buchstabe einer Prozedur
; gross geschrieben, wenn es sich nicht um
; eine Standardprozedur handelt.
; default: lower
;*****
;SUBPROCPREFIX: */xyz
; Prefix zu Subprozeduren. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: F
;*****
;PROCVARPREFIX: */xyz
; Prefix zu Prozedurvariablen. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 'V'
;*****
;RESERVEDPREFIX: */xyz
; Prefix zu Objekten, deren Namen in C reserviert
; sind (Bsp: Variable EXTERN in TAL wuerde zu extern
; in C, was zu Fehler fuehrt.). Es wird RESERVEDPREFIX
; vorangestellt. Hier ist * wirkungslos, es muss
; in jedem Fall ein prefix angegeben sein.
; default: 'S'
;*****
;TYPEDEFPREFIX: */xyz
; Prefix zu Strukturtypen. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 'T'
;*****
;INFVOKEPREFIX: */xyz
; Prefix zu Strukturtypen, die aus INVOKE
; entstanden sind. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 't'
;*****
;SHADOWPARMPREFIX: */xyz
; Prefix zu Parametern in Subprozeduren. Fuer den

```

```

; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 'H'
;*****
;ENTRYPROCPREFIX: */xyz
; Prefix zu ENTRY-Funktionen. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 'Y'
;*****
;PARMASSIGNPREFIX: */xyz
; Prefix zu Indikatorvariablen, welche die $PARAM-Funktion
; fuer Parameter simulieren. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 'A'
;*****
;TEMPVARPREFIX: */xyz
; Prefix zu Translatorvariablen. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 'E'
;*****
;ANONYMUNIONPREFIX: */xyz
; Prefix zu anonymen unions. Fuer den
; Fall, dass ein * angegeben ist, entfaellt
; dieser Prefix.
; Achtung: * kann zu Namenskonflikten fuehren.
; default: 'u_'
;*****
;
;STRUKTURINITIALISIERUNG
;
;*****
;DIRECTINITIAL: n/automatic
; Es wird die Generierung von Initialisierungen
; wie folgt modifiziert:
; Strukturen, die initialisierte Felder ueberlagern,
; werden direkt mit den Initialwerten des Feldes
; initialisiert, das Feld selbst wird nicht erzeugt.
; Dabei gelten folgende Werte:
; n: Wie gehabt, beide werden generiert.
; automatic: Die Initialisierung wird durch
; den Translator so vorgenommen, dass die
; bestmoegliche Form realisiert wird.
; default: n
;*****
;*****
;DIRECTINITIAL_STRING: y/n
; Wenn auf y gesetzt, so wird in der direkten
; Stringinitialisierung die Erzeugung von
; Zeichenketten statt Zeichenfeldern erzeugt.
; default: n
;*****
;*****
;DIRECTINITIAL_REDEFINE: y/n
; Wenn auf y gesetzt, so werden Strukturen auch
; dann direkt initialisiert, wenn redefine-Komponenten
; darin enthalten sind.
; default: n
;*****
;*****
;ZERTEILUNGSINFORMATIONEN
;
;*****
;RECURSIVE_INCLUDE: y/n
; Fuer den Fall, dass y gesetzt, werden
; alle DIVSTART/DIVEND-Direktiven rekursiv

```

```

; ausgegeben, d.h., es wird vor jedem zu zerteilenden
; Datum die vollstaendige include-Hierarchie
; geschrieben
; (Diese Option ist vorerst ausgeklammert, da sie
; fuer die Zerteilung zwangsweise benoetigt wird.
; Das Setzen/Nichtsetzen hat damit auf die
; Abarbeitung keinerlei Einfluss.).
; default: y
;*****
;SUPPRESS_INIT_SPLIT: y/n
; Wenn auf y gesetzt, wird die
; bei der Generierung von Zerteilerinformationen
; entstehende Aufsplittung von _Ar_init fuer jedes
; Modul unterdrueckt und alle Ergebnisse in ein
;_Ar_init geschrieben.
; default: n
;*****
;INSERT_DIVIDE_INFO: y/n
; Wenn auf y gesetzt, werden die Zerteilungsinformationen
; eingefuegt (identisch zu -div).
; default: n
;*****
;STRUCT_DIVIDE: y/y_and_debug/n
; Spezielle Loesung fuer HDM:
; Wenn auf y gesetzt, werden Strukturen der Form:
; STRUCT X;
; ?SOURCE AAA
; so zerteilt, dass sie in AAA landen. Hat eine
; Struktur mehr als eine Source
; STRUCT X;
; ?SOURCE AAA
; ?SOURCE BBB
; so bleibt alles in der Source von X. Wenn auf
; y_and_debug gesetzt, so werden zusaetzlich
; Informationen ueber den Berechnungsprozess ausgegeben.
; default: n
;*****
;
;SONSTIGES
;
;*****
;LITERAL: upper/lower
; Buchstaben in Literalen werden
; gross/klein geschrieben.
; default: lower
;*****
;PRAGMA_PAGE: y/n
; Wenn nicht angegeben, werden alle
; #pragma page generiert, bei n nicht.
; default: y.
;*****
;PRAGMA_LIST: y/n
; Wenn nicht angegeben, werden alle
; #pragma [no]list generiert, bei n nicht.
; default: y.
;*****
;INSERT_FILE_LOCATION: y/n
; Falls auf y gesetzt, wird vor jede globale
; Definition (BLOCK,Daten,Funktionen,...) deren
; Filelokalisierung geschrieben.
; default: y
;*****
;WRITE_BUFFERED: y/n
; Falls auf y gesetzt, erfolgt die gesamte
; Generierung zunaechst auf einen internen Puffer
; (set_buffered_access), erst danach wird die Geschichte
; ausgeschrieben. Fuer den Fall Windows oder BORLANDC/C_ONLY
; wird in jedem Fall gepuffert gearbeitet,
; da BORLAND die fseek/ftell-Geschichte nicht
; korrekt beherrscht.
; default: n

```

```
;*****  
;BLOCKINDENT: style-1/style-2  
; Bei style-2 werden HDM-Wuensche wirksam:  
; - keine Einrueckung bei Prozedurbeginn  
; - Blockklammern auf gleicher Hoehe  
; wie die Anweisungen darin.  
; default: style-  
;*****
```

8 Das File tal.stp

8.1 Funktionalität des Files tal.stp

Das File `tal.stp` ist Bestandteil des Translatorkpaketes und ein ASCII-File. Die Nutzung des Files `tal.stp` erfolgt aus drei Gründen:

1. Bekanntmachen der Guardian Calls im *cg*n :
Es sind alle Funktionsnamen, die als Guardian Calls bekannt sind, in einer speziellen Syntax enthalten.
2. Definition von Vorgaben (Signaturen) für Prozeduren:
Es ist möglich, bei der Konvertierung von TAL-Prozeduren und -Subprozeduren nach C/C++ den C-Typ der Parameter und des Rückgabewertes vorzugeben. Solche Vorgaben werden in einer speziellen Sektion des Files `tal.stp` definiert.
3. Bekanntmachen der TAL-Standardfunktionen in *cg*n :
TAL besitzt eine Reihe von Standardfunktionen, die verschiedenartig parametrisiert sein können. Die Deklaration der Standardfunktionen für den *cg*n erfolgt über das File `tal.stp` in einer definierten Form.

8.2 Syntax und Semantik des Files tal.stp

Die Syntax des Files wird in BNF (Backus-Naur-Form) beschrieben:

```
setup_file:
guardian_call_section
c_signatur_section
stdfun_section
;

/* Definition der Guardian-Systemrufe */
guardian_call_section:
~BEGIN_Guardian_CALL
LIST OF guardian_call
~END_Guardian_CALL
;

guardian_call:
identifier
| ~CC_STATUS identifier
;

/* Vorgabe der Typen bei der Konvertierung von Prozeduren */
c_signatur_section:
~BEGIN_PROTOTYPES
LIST OF c_prototype
~END_PROTOTYPES
| /* kann fehlen */
;

c_prototype:
descriptor ':' c_function ';'
;

descriptor:
identifier
| identifier '.' identifier
;

c_function:
c_type c_identifier c_parmlist
;
```

```

c_type:
c_basic_type
| c_function_type
;

c_basic_type:
c_signed_extension 'char' c_operator_extension
| c_signed_extension 'int' c_operator_extension
| c_signed_extension 'short' c_operator_extension
| c_signed_extension 'long' c_operator_extension
| 'long' 'long' c_operator_extension
| 'float' c_operator_extension
| 'double' c_operator_extension
| 'void' c_operator_extension
| 'struct' c_identifier c_operator_extension
| 'union' c_identifier c_operator_extension
;

c_function_type:
c_basic_type '(' '*' ')' c_parmlist
;

c_signed_extension:
'signed'
| 'unsigned'
| /* kann leer sein */
;

c_operator_extension:
'*'
| '*' '*'
| /* kann leer sein */
;

c_parmlist:
 '(' ')' /* leere Parameterliste */
| '(' c_typliste ')'
;

c_typliste:
c_type
| c_type ',' c_typliste
;

c_identifier: /* ein konventioneller C--Bezeichner */
;
/* Definition der TAL-Standardfunktionen */
stdfunc_section:
LIST OF stdfunc
'~%%~'
;

stdfunc:
~PROC set-name interface parmlist returntyp ';'
| ~PROC set-name ~OVL nextname interface parmlist returntyp ';'
;

parmlist:
 '(' !tal-aeahnliche Liste formaler Parametertypen? | ~GENERIC ')'
| /* kann leer sein */
;

interface:
 '[' tal-name ',' /* Name der Funktion in TAL */
set-name ',' /* Name der Funktion in tal.stp */
impl-name ',' /* Name der implementierten Funktion */
mtyyp ',' /* Art der Standardfunktion */
 '(' parmtmpl ')' /* Parametertemplate */
']'
;

```

```

returntyp :
'(' !typischer TAL-Rueckkehrtyp? ') '
| /* kann leer sein */
;

parmtmpl :
identifizier
| /* kann leer sein */
;

nextname:
identifizier
;

set-name:
identifizier
;

tal-name:
identifizier
;

impl-name:
identifizier
;

mtyp:
~SIMPLE_STDFUNC
| ~PARAM_STDFUNC
;

identifizier: /* ein konventioneller TAL--Bezeichner */
;

```

In dieser Syntax sind

```

~OVL, ~SIMPLE_STDFUNC, ~PROC
~PARAM_STDFUNC, ~GENERIC, ~CC_STATUS
~BEGIN_Guardian_CALL, ~END_Guardian_CALL

```

sowie alle in doppelte Hochkommas eingeschlossenen Worte (z.B. "char") Schlüsselworte. An den Stellen, in denen Kommentare der Form < : : > auftreten, werden TAL- bzw. C-typische Konstrukte eingesetzt. `identifizier` ist ein konventioneller TAL-Bezeichner und `c_identifizier` ist ein Bezeichner, welcher den C-Konventionen genügt. In der `guardian_call_section` und der `stdfunc_section` sind alle Schlüsselworte und Bezeichner in Großbuchstaben zu schreiben. Die Formulierungen

```

LIST OF stdfunc
LIST OF guardian_call
LIST OF c_prototype

```

besagen, daß eine Liste über `stdfunc`, `guardian_call` bzw. `c_prototype` gebildet werden kann. Das File `tal.stp` muß mit der Zeichenfolge `~%%~` abgeschlossen werden. Kommentare beginnen mit der Zeichenfolge `~@` und enden mit dem folgenden Zeilenende. Dieser Grammatik unterliegt die nachfolgende Bedeutung:

1. Semantik der `guardian_call_section`
Der erste Teil der Definitionen bezieht sich auf Guardian Calls. Es werden die Namen der Guardian Calls notiert. Setzen diese das Condition-Register, so ist das Schlüsselwort `~CC_STATUS` vorangestellt.
2. Semantik der `c_signatur_section`
Der zweite Abschnitt des Files `tal.stp` ist, wie der Syntaxbeschreibung entnommen werden kann, optional und kann auch vollständig (d.h. einschließlich `~BEGIN_PROTOTYPES` und `~END_PROTOTYPES`) weggelassen werden. Ist die Klammerung `~BEGIN_PROTOTYPES : : : ~END_PROTOTYPES`

jedoch vorkanden, so muß sie wenigstens einen Eintrag enthalten. Im weiteren soll die Bedeutung dieser Einträge erläutert werden. Jeder Eintrag beginnt mit einem Deskriptor, welcher für diesen Eintrag eindeutig sein muß. Dabei gibt es zwei verschiedene Arten von Deskriptoren:

- `identifizier`:
Hierbei ist `identifizier` genau der Name einer TAL-Top-Prozedur. Damit kann die Übersetzung von TAL-Prozeduren gesteuert werden.
- `identifizier1.identifizier2`:
Hierbei ist `identifizier1` der Name einer Top-Prozedur und `identifizier2` der Name einer in dieser Top-Prozedur definierten Sub-Prozedur. Damit kann die Übersetzung von TAL-Sub-Prozeduren gesteuert werden.

Bei der Übersetzung einer TAL-Prozedur (Top- oder Sub-Prozedur) durchsucht der TTC zunächst diese Liste. Wird ein Eintrag gefunden, bei dem der Prozedurname (bzw. der Top- und Sub-Prozedurname bei Sub-Prozeduren) mit dem Namen (bzw. Top- und Sub-Prozedurnamen) der aktuell zu übersetzenden TAL-Prozedur übereinstimmt, so werden durch diesen Eintrag die Parametertypen und der Rückgabotyp festgelegt.

Wird kein entsprechender Eintrag gefunden, so berechnet der TTC selbst diese Typen aus den TAL-Originalen. Es werden dabei die gleichen Regeln zugrundegelegt, wie bei der Umsetzung von Variablen.

Nach dem Deskriptor steht die eigentliche Beschreibung des C-Funktions-Prototypes (`c_function`). Die Syntax dieser Beschreibung ist an die Syntax von C-Funktionsköpfen angelehnt.

Allgemein besteht sie aus drei Teilen:

- (a) `c_type`:
Das ist der Rückgabotyp der Funktion. Handelt es sich um eine Prozedur, die nichts zurückgibt, so ist hier `void` zu schreiben.
- (b) `c_identifizier`:
Das ist der Name, den die Funktion in C erhalten soll. Die translatorinterne Namensvergabe wird für diese Funktion abgeschaltet.
- (c) `(' c_typliste ')` bzw. `(' ')'`:
Das ist die Liste der Parametertypen. Hat die Funktion keine Parameter, so muß `(' ')'` angegeben werden. Anderenfalls steht zwischen den Klammern eine, durch Komma getrennte, Liste von C-Typen. Anders als bei der Definition von C-Funktionsköpfen werden hier die Namen der Parameter angegeben.

Achtung: Die Anzahl der Parametertypen in dieser Liste muß mit der Anzahl der Parameter der TAL-Prozedur bzw. -Sub-Prozedur übereinstimmen. Sonst wird eine Fehlermeldung ausgegeben.

Die Semantik der möglichen C-Typen kann der Literatur entnommen werden. Sie soll hier nicht im einzelnen wiederholt werden. Wichtig ist, daß nicht alle in C möglichen Typen angegeben werden können. Felder, `enum`-Typen und selbstdefinierte Typbezeichner (`typedef`) sind nicht möglich. Bei Strukturen und `unions` muß genau der Name verwendet werden, welcher vom TTC für diese Struktur vergeben wird. Auch die Angabe der Speicherklasse `register` ist nicht gestattet.

Der `i`: Parameter der umgesetzten Prozedur erhält genau den Typ in C, welcher an der `i`: Stelle in der C-Typliste enthalten ist. Bei der Verwendung des Parameters in Ausdrücken innerhalb des Funktionsrumpfes können dann entsprechende `cast`-Operatoren nötig sein. Probleme können auftreten, wenn ein entsprechender `cast` in C nicht definiert ist. Wenn beispielsweise in TAL der Parameter den Typ `INT` besitzt und in der C-Typenliste wird ein Strukturtyp vorgegeben, so wird bei der Verwendung des Parameters in einem arithmetischen Ausdruck eine Fehlermeldung vom TTC ausgegeben, da in C kein `cast` von einem Strukturtyp

in einen Zahlen-Typ definiert ist. Analog werden beim Aufruf einer Funktion mit vorgegebenem C-Typ die aktuellen Parameter in den (vorgegebenen) C-Typ gecastet. Dabei können ebenfalls Fehler aufgrund von nichtdefinierten cast-Operatoren auftreten. cast-Operatoren werden natürlich nur dort geschrieben, wo sie in C wirklich nötig sind.

Daraus geht hervor, daß die Wahl der zu vergebenden C-Typen nicht völlig frei ist. Sie hängt vielmehr von der Verwendung der Parameter und der Funktion in Ausdrücken ab. Wird z.B. die Funktion innerhalb einer arithmetischen Operation verwendet, so kann der Rückgabewert keine Struktur sein.

Enthält eine TAL-Prozedur mit vorgegebenen Typ "Entries" (alternative Einsprungpunkte), so erhalten die "Entries" die gleichen Parametertypen und den gleichen Rückgabotyp wie die Prozedur selbst.

Am Ende dieses Abschnittes soll ein Anwendungsbeispiel die genannte Möglichkeit demonstrieren. Es zeigt, wie das Problem der Umsetzung von Prozeduren, bei denen eine Adresse als INT-Wert übergeben wird, gelöst werden kann:

TAL-Prozedur:

```
INT PROC INC(ADR, Y);
INT ADR; !Adresse des zu erh'ohenden INT-Wertes!
INT Y; !Wert um den erh'ohet wird!
BEGIN
  .ADR := .ADR+Y;
  RETURN .ADR;
END;
...
INT X;
...
IF (INC(@X,7) < 100) THEN
  ...
```

Zunächst die Konvertierung ohne Typanpassung im File tal.stp.

```
short inc (short adr, short y)
{
  *((short*)(long)adr) = *((short*)(long)adr)+y;
  return *((short*)(long)adr);
}
...
short x;
...
if ( inc(ADR2SHORT__(&x),7) < 100 )
  ...
```

Nun wird die folgende Zeile in das File tal.stp eingefügt:

```
INC : short inc(short*, short);
```

Das bewirkt, daß das konvertierte C/C++-Programm jetzt wie folgt aussieht:

```
short inc(short *adr, short y)
{
  *adr = *adr+y;
  return *adr;
}
...
short x;
...
if ( inc(&x,7) < 100 )
  ...
```

In diesem Fall wird ein (fehleranfälliger) cast von short* nach short durch ADR2SHORT__() eliminiert.

3. Semantik der `stdfun_section`

Im dritten Teil werden die Standardfunktionen beschrieben. Dabei kann jede Standardfunktion mehrfach auftreten, indem sie mit verschiedenen Parameterlisten formuliert wird. Diese Methodik sichert die Möglichkeit der Überladung einer Funktion für mehrere Parametertypen. Generell werden alle Funktionen, die eine Standardfunktion beschreiben, in einer logischen, linearen Liste verkettet. Diese lineare Liste wird über den `~OVL` Mechanismus nachgebildet. Das folgende Beispiel demonstriert eine solche Verkettung für die Standardfunktion `$ABS` in TAL:

```

~@ procedure 72
~PROC $ABS_i
~OVL $ABS_I
[$ABS, $ABS_i, c_abs_i, ~PARAM_STDFUNC, (i)]
(INT)
~RETURNS (INT);

~@ procedure 72
~PROC $ABS_I
~OVL $ABS_r
[$ABS, $ABS_I, c_abs_I, ~PARAM_STDFUNC, (I)]
(INT(32))
~RETURNS (INT(32));

~@ procedure 72
~PROC $ABS_r
~OVL $ABS_R
[$ABS, $ABS_r, c_abs_r, ~PARAM_STDFUNC, (r)]
(REAL)
~RETURNS (REAL);

~@ procedure 72
~PROC $ABS_R
~OVL $ABS_U
[$ABS, $ABS_R, c_abs_R, ~PARAM_STDFUNC, (R)]
(REAL(64))
~RETURNS (REAL(64));

~@ procedure 72
~PROC $ABS_U
~OVL $ABS_S
[$ABS, $ABS_U, c_abs_U, ~PARAM_STDFUNC, (U)]
(UNSIGNED)
~RETURNS (UNSIGNED);

~@ procedure 72
~PROC $ABS_S
~OVL $ABS_F
[$ABS, $ABS_S, c_abs_S, ~PARAM_STDFUNC, (S)]
(STRING)
~RETURNS (STRING);

~@ procedure 72
~PROC $ABS_F
[$ABS, $ABS_F, c_abs_F, ~PARAM_STDFUNC, (F)]
(FIXED)
~RETURNS (FIXED);

```

Dabei ist die erste Funktion (`ABS_i`) jene, die für den Parametertyp `INT` zugelassen ist. Alle anderen Parametertypen dürfen für diesen Fall nicht auftreten. Damit bildet `ABS_i` eine Einschränkung der allgemeinen `$ABS`-Funktion hinsichtlich der zugelassenen Parametertypen. Ein Parameter `FIXED` wäre hier beispielsweise nicht möglich. Um die Garantie zu bieten, daß die Funktion universell für alle Typen zugelassen ist, wird eine "Nachfolgefunktion" aufgebaut. Diese ist mit dem Namen `ABS_I` versehen. Die Verkettung wird in der `ABS_i` Deklaration durch die `~OVL` Methodik angelegt (`OVL` steht für `overload`). Nach dem Schlüsselwort `~OVL` folgt der Name der Funktion, die für weitere Parametertypen zulässig ist. Diese ist wiederum als `~OVL` definiert. `ABS_F` ist ohne das Attribut `~OVL` definiert, d.h. diese Funktion ist der Abschluß dieser linearen Liste. Für die Liste gelten folgende Festlegungen:

- (a) Die Listenelemente stehen im File `tal.stp` direkt hintereinander.
- (b) Die Liste kann beliebig lang sein.
- (c) Hat die Liste eine Länge von n Elementen, dann besitzen die ersten $n-1$ Elemente das Attribut `~OVL`, die n . Funktion besitzt dieses Attribut nicht (Listenabschluß).
- (d) Innerhalb einer Liste erfolgt die Verkettung mit der nächsten Funktion durch Nutzung des `~OVL` Mechanismus. Hinter `~OVL` muß der Name der Funktion angegeben werden, die als folgende in der Liste auftritt. Im Beispiel folgt auf `ABS_U` durch die Angabe `~OVL $ABS_S` die Funktion `$ABS_S`.
- (e) Die Namen der Funktionen sind wahlfrei nach den Konventionen von TAL (incl. erstem Unterstrich) zu bilden.

Die Art und Weise, wie die Funktion während der Übersetzungszeit zu behandeln ist, wird in der Zeile zu interface beschrieben. Bezüglich obiger Grammatik gelten dabei folgende Richtlinien:

- (a) `set-name` ist der Name der Funktion in `tal.stp`, er ist identisch zum Namen nach dem PROC. Diese Notation ist zwar redundant, für die effektive Translationsarbeit aber von Notwendigkeit.
- (b) `tal-name` ist der Name der Funktion, wie sie in TAL benutzt wird.
- (c) `impl-name` ist der Name der implementierten Funktion. In der vorliegenden Version sind alle Standardfunktionen in C bzw. C++ implementiert und unter entsprechenden Namen deklariert. Diese Deklarationen werden während der Übersetzung des generierten C/C++-Files geincludet. Im obigen Beispiel ist `c_abs_I` der Name der Funktion, die die Berechnung des Absolutbetrages für `INT(32)`-Typen vornimmt. Diese Funktion wird vom `cgn` generiert.
- (d) `mtyp` ist eine spezielle Bezeichnung dafür, nach welchen Kriterien die Auswahl der konkreten Funktion aus der Gesamtliste vorgenommen werden soll. Dabei gilt, daß alle Funktionen einer Liste mit ein und demselben `mtyp` ausgerüstet werden müssen. Wird dieser Fakt unterlassen, so ist die Wirkung ungewiß. Für `mtyp` können dabei folgende Werte auftreten:
 - `~SIMPLE_STDFUNC` tritt immer und nur dann auf, wenn die Liste der Funktionen die Länge 1 hat, also `~OVL`-Attribute nicht auftreten. In diesem Fall hat der `cgn` keine Wahl, er muß in jedem Fall genau die eine Funktion generieren. Für `~SIMPLE_STDFUNC` ist auch `parmtmpl` immer leer.
 - `~PARAM_STDFUNC` kann nur in einer Liste auftreten, deren Länge größer als 1 ist. In diesem Fall sind alle formalen Parameterlisten von gleicher Länge. Die Auswahl der konkreten Funktion aus der Liste erfolgt ausschließlich nach den angegebenen formalen Parametertypen. Diese Auswahl allerdings wird nicht anhand von `parmlist` getroffen, sondern anhand von `parmtmpl`.

Dabei wird folgender Auswahlalgorithmus angewendet:

```
>> sei l die Liste der tal.stp-Funktionen zur
Standardfunktion s <<
>> sei k die erste Funktion der Liste <<
>> solange noch keine passende Funktion gefunden<<
    >> wenn die Parametertypen der aktuellen Funktion typgleich
        zu den Parametertypen in parmtmpl sind<<
        >>generiere diese Funktion, sie wurde gefunden.<<
    >> andernfalls <<
        >> wenn k die letzte Funktion ist <<
            >>generiere diese Funktion, sie wurde gefunden.<<
        >> andernfalls <<
            >> k:= folgelement(k) in l<<
        end /* wenn */
    end /* wenn */
end /* solange */
```

In `parmtmpl` steht je ein Buchstabe aus `{A..Z; a..z}` für einen nutzbaren Typ. Aus der Stellung des Buchstabens innerhalb von `parmtmpl` wird auf die Stellung des Parameters in der formalen Parameterliste geschlossen, d.h. der erste Buchstabe in `parmtmpl` steht für den ersten Parameter, der zweite für den zweiten usw. Im einzelnen unterliegt den einzelnen Buchstaben folgende Typbedeutung:

Kürzel und ihre Typen:

Kürzel	Typ
i	INT
I	INT(32)
r	REAL
R	REAL(64)
U	UNSIGNED
S	STRING
F	FIXED
G	~GENERIC, d.h. ein beliebiger Typ
V	VOID, d.h. kein signifikanter Typ
e	INT, STRING oder UNSIGNED
E	INT(32) oder UNSIGNED
Z	INT, FIXED, REAL oder REAL(64)
Y	INT, INT(32), FIXED oder REAL
X	INT, INT(32), FIXED oder REAL(64)
A	INT, INT(32), FIXED, REAL oder REAL(64)
B	INT(32), FIXED, REAL, REAL(64)

`parmtmpl` wurde bereits im Anstrich zu `mtyp` ausführlich behandelt.

- (e) Rückkehrtypen und Parameterlisten sind vollständig zu beschreiben. In den Parameterlisten kann zusätzlich das Schlüsselwort `~GENERIC` mit oben genannter Bedeutung auftreten.

Hinweis:

Indem ein TAL-Makro (`DEFINE`) genau wie eine Standardfunktion in diese Sektion eingetragen wird, kann die Expandierung der TAL-Defines verhindert werden. Das funktioniert natürlich nur, wenn der Makroaufruf wie ein korrekter TAL-Funktionsaufruf (natürlich ohne `CALL`) aussieht. Ansonsten generiert der TTC einen Syntaxfehler!

In diesem Fall wird wieder ein C-Funktionsruf (entsprechend den Angaben aus dem File `tal.stp`) generiert. Die Implementierung der Funktion (evtl. auch als C-Makro) obliegt dem Nutzer, sofern es sich nicht um eine C-Standardfunktion oder C-Bibliotheksfunktion handelt. Der Makrokörper wird nicht automatisch konvertiert.

Beispiel:

Der Eintrag

```
~PROC ^LEN_G  
  [ ^LEN, ^LEN_G, sizeof, ~SIMPLE_STDFUNC, (G) ]  
  (~GENERIC)  
  ~RETURNS (INT);
```

bewirkt, daß der Aufruf

```
^LEN(X)
```

als

```
sizeof(x)
```

übersetzt wird.

Das File `tal.stp` wird vor der Bearbeitung des eigentlichen TAL-Files analysiert. Treten hier bereits Fehler auf, wird die Konvertierung abgebrochen.

Anhang A: Fehler und Warnungen *tpp* und *tal_only*

A.1 Fehlermeldungen des *tpp*

Nummer : X100
Meldung : Argument bei Option `-r` fehlt.

Nummer : X101
Meldung : Argument bei Option `-c` fehlt.

Nummer : X102
Meldung : Option `-H` braucht kein Argument

Nummer : X103
Meldung : Argument bei Option `-I` fehlt

Nummer : X104
Meldung : Argument bei Option `-D` fehlt

Nummer : X105
Meldung : Argument bei Option `-o` fehlt

Nummer : X106
Meldung : unbekannte Option

Nummer : X107
Meldung : kein Filename angegeben

Nummer : X108
Meldung : File `<filename>` kann nicht geöffnet werden
Bedeutung : Das File `<filename>` wird zwar gefunden, kann aber nicht zum Lesen geöffnet werden.
Meist sind hier die Zugriffsrechte nicht richtig gesetzt.

Nummer : X109
Meldung : Zu dem `IF/IFNOT` in Zeile `<nr>` des Files `<filename>` gibt es kein `ENDIF` in der gleichen `SECTION <sectionname>`
Bedeutung : Zu jedem `?IF` oder `?IFNOT` muß in der gleichen `SECTION` (bzw. im gleichen File, wenn es nicht in `SECTION` zerteilt ist) ein korrespondierendes `?ENDIF` stehen.

Nummer : X110
Meldung : `<filename>` kann nicht gefunden werden
Bedeutung : Das File `<filename>` wird in keinem der durchsuchten Verzeichnisse gefunden.
Entweder liegt ein Schreibfehler des Filenamens vor oder das Verzeichnis mit dem File wird nicht durchsucht.

Nummer : X111

Meldung : <filename> kann nicht geöffnet werden

Bedeutung : Das File <filename> wird zwar gefunden, kann aber nicht zum Lesen geöffnet werden. Meist sind hier die Zugriffsrechte nicht richtig gesetzt.

Nummer : X112

Meldung : interner Fehler: File <filename> nicht in der Bibliothek

Bedeutung : Interner Fehler im Präprozessor. Bitte setzen Sie sich mit pro et con in Verbindung.

Nummer : X113

Meldung : Benutzung: tpp opt_1 ... opt_n <fname>

Nummer : X114

Meldung : Interner Fehler Nr. <nr>

Bedeutung : Interner Fehler im Präprozessor. Bitte setzen Sie sich mit pro et con in Verbindung

Nummer : X115

Meldung : Unterminierter String

Bedeutung : Ein in einer Zeile begonnener String () wurde nicht in derselben Zeile mit abgeschlossen.

Nummer : X116

Meldung : SECTION <filename> im File <sectionname> nicht gefunden

Nummer : X117

Meldung : Syntaxfehler in Direktive

Nummer : X119

Meldung : ENDIF hat kein korrespondierendes IF

Bedeutung : Es wurde ein ?ENDIF gefunden, aber in derselben SECTION steht kein zugehöriges ?IF oder ?IFNOT.

Nummer : X121

Meldung : File <filename> kann nicht wieder geöffnet werden

Bedeutung : Das File wurde einmal gefunden (z.B. beim Lesen der ersten SECTION). Beim Versuch, es zum Zwecke des Lesens einer weiteren SECTION wieder zu öffnen, trat ein Fehler auf. Der Grund ist, daß das File in dieser Zwischenzeit gelöscht wurde.

Nummer : X122

Meldung : File <filename> SECTION <sectionname> zieht sich selbst ein

Bedeutung : Eine SECTION enthält eine ?SOURCE-Direktive, welche dieselbe SECTION referenziert. Das ist in TAL nicht erlaubt.

Nummer : X123

Meldung : File <filename> zieht sich selbst ein

Bedeutung : Ein File enthält eine ?SOURCE-Direktive, welche dasselbe File referenziert. Das ist in TAL nicht erlaubt.

A.2 Fehlermeldungen des Programmes *tal_only*

Nummer : X124

Meldung : Benutzung: *tal_only* <filename>.tln

Nummer : X125

Meldung : File <filename> kann nicht zum Lesen geoeffnet werden

Nummer : X126

Meldung : File <filename> kann nicht erzeugt werden

Nummer : X127

Meldung : gleicher Name (<filename>) fuer Ausgabe- und Kommentarfile ist verboten

Anhang B: Fehler,Warnungen und Sorrrys des *cg*

In diesem Abschnitt erfolgt die Auflistung aller Meldungen, die durch den *cg* erzeugt werden. Dabei gelten folgende Richtlinien:

- "%s" ist ein Platzhalter für eine Zeichenkette, die in der konkreten Meldung an dieser Stelle eingesetzt wird.
- "%d" ist ein Platzhalter für eine Zahl, die in der konkreten Meldung an dieser Stelle eingesetzt wird.
- Die Meldungen sind alphabetisch und nach Nummern sortiert.

Nummer: D001
Art: Fehler
Meldung: Interner Fehler in %s: Objektzeiger auf %s unzulaessig. %s erwartet

Nummer: D002
Art: Fehler
Meldung: Interner Fehler in %s: Objektzeiger auf %s unzulaessig. %s oder %s erwartet

Nummer: D003
Art: Fehler
Meldung: Interner Fehler in %s: Objektzeiger auf %s unzulaessig. %s, %s oder %s erwartet

Nummer: D004
Art: Fehler
Meldung: Interner Fehler in %s: TAL-Typ %s ist hier nicht zulaessig

Nummer: D201
Art: Warnung
Meldung: Interne Warnung in %s: Objektzeiger auf %s wird nicht behandelt

Nummer: D202
Art: Warnung
Meldung: Interne Warnung in %s: TAL-Typ %s wird nicht behandelt

Nummer: D203
Art: Warnung
Meldung: Interne Warnung in %s: Basisvariable %s faelschlich als Ueberlagerung gekennzeichnet

Nummer: D204
Art: Warnung
Meldung: Interne Warnung in %s: Ueberlagerung %s nicht als solche gekennzeichnet

Nummer: D205
Art: Warnung
Meldung: Interne Warnung in %s: Identifikator %s als NONE OVL gekennzeichnet

Nummer: D401
Art: Fehler

Meldung: Stringliteral %s muss zwischen 1 und 4 Zeichen enthalten
Bedeutung: Literaldefinitionen, deren Werte Zeichenketten sind, dürfen maximal 4 Zeichen enthalten.

Nummer: D402

Art: Fehler

Meldung: Nicht portable Ueberlagerung des indirekten Datums %s durch den Skalar %s

Bedeutung: In TAL wird nicht das indirekte Datum selbst, sondern der implizite Zeiger darauf überlagert. Diese lt. TAL-Reference-Manual nicht empfohlene Programmierweise kann in C/C++ nicht nachgebildet werden.

Nummer: D602

Art: Warnung

Meldung: Ueberlagerung des indirekten Datums %s durch den Zeiger %s kann zu Laufzeitfehlern fuehren

Bedeutung: Die semantische Äquivalenz des Programmes ist nur gegeben, solange der Zeiger selbst nicht verändert wird.

Nummer: D801

Art: Sorry

Meldung: Speicherplatzloses Feld %s wird nicht behandelt

Bedeutung: Felder, die in TAL keinen Speicherplatz besitzen, werden nicht konvertiert.

Nummer: F208

Art: Warnung

Meldung: Datenfluss von %s nach %s ignoriert

Bedeutung: Der Name eines Datums tritt in einer Datendefinition sowohl auf der rechten als auch auf der linken Seite der Definition auf. Die daraus resultierende Abhängigkeit wird in der Datenflußanalyse nicht berücksichtigt.

Beispiel:

```
STRING FELD[0:10] := $LEN(FELD) * [ ];
```

Nummer: F206

Art: Fehler

Meldung: Programm enthaelt zyklische Ueberlagerungen

Bedeutung: Im Programm treten zyklische Überlagerungen auf, eine Speicherplatzberechnung ist damit nicht korrekt möglich.

Nummer: G088

Art: Warnung

Meldung: Kann Multiplikator nicht berechnen

Bedeutung: In einer Initialisierungsliste tritt ein Wiederholungsfaktor auf, dessen Wert nicht berechnet werden kann, d.h. in diesem Ausdruck sind variable Komponenten enthalten. Der erzeugte C/C++-Code kann dennoch korrekt sein.

Nummer: G089

Art: Warnung

Meldung: Kann Initialwert nicht berechnen

Bedeutung: In einer Initialisierungsliste tritt ein Ausdruck auf, dessen Wert nicht berechnet werden kann, d.h. in diesem Ausdruck sind variable Komponenten enthalten. Der erzeugte C/C++-Code kann dennoch korrekt sein.

Nummer: G140

Art: Sorry

Meldung: %s nicht implementiert

Bedeutung: Die genannte Anweisung ist in der vorliegenden Version des Translators nicht implementiert. Insbesondere werden hier maschinenabhängige Befehle wie `USE`, `STACK` oder `CODE` auftreten.

Nummer: G150

Art: Warnung

Meldung: CC-Register in %s nicht gesetzt

Bedeutung: Es erfolgt kein Setzen des Conditioncode-Wertes in der genannten Operation.

Nummer: G152

Art: Warnung

Meldung: Funktion %s ohne `RETURN` (eingefuegt)

Bedeutung: Eine Funktionsprozedur besitzt keinen Returnwert in der genannten Operation.

Nummer: G320

Art: Fehler

Meldung: `INT(32)`-Typ nicht erlaubt

Bedeutung: An dieser Stelle des Ausdrucks darf kein `INT(32)`-Typ auftreten.

Nummer: G321

Art: Fehler

Meldung: `STRING` oder `INT` Typ erwartet

Bedeutung: An dieser Stelle des Ausdrucks wird ein `STRING` oder ein `INT`-Typ erwartet.

Nummer: G400

Art: Warnung

Meldung: `GUARDIAN` Funktion %s redefiniert

Bedeutung: Innerhalb des Programmes wurde eine Prozedur definiert, die namensgleich mit einer `Guardian`-Funktion ist.

Nummer: G500

Art: Fehler

Meldung: Fehlerhafter Typ im `MOVE`-Statement

Bedeutung: Innerhalb eines `MOVE`-Statements trat ein Typ auf, der nicht zugelassen ist.

Nummer: G506

Art: Warnung

Meldung: `INT`-Feld fuer direkte Strukturinitialisierung

Bedeutung: Es können Ausrichtungsfehler auftreten, wenn bei Überlagerung eines `INT`-Feldes mit einer Struktur eine direkte Strukturinitialisierung `DIRECTINITIAL_REDEFINE` versucht wird.

Nummer: G544

Art: Warnung

Meldung: Literal %s expandiert

Bedeutung: Das genannte Literal wird an dieser Stelle durch seinen Wert ersetzt (expandiert). Diese Expandierung ist insbesondere dann notwendig, wenn das Literal in TAL eine Zeichenkette darstellt. Diese wird in C/C++ durch einen ganzzahligen Wert repräsentiert. Soll das Literal beispielsweise in einer `MOVE`-Anweisung benutzt werden, so wird dort dessen Zeichenkettenwert erwartet.

Nummer: G588

Art: Warnung

Meldung: Zum Linken des Moduls expliziten Ruf der Funktion `_Ar_init_xxx` erzeugen

Bedeutung: Die in diesem Modul enthaltene Funktion `_Ar_init_xxx` muß entweder durch Nutzung des Werkzeuges `itc` oder durch ein handgeschriebenes Programm gerufen werden.

Nummer: G622

Art: Warnung

Meldung: Option `-div` nur auf `tp` bearbeitete Files anwendbar

Bedeutung: Option `-div` nur auf Files anwenden, die vom Präprozessor erstellt wurden.

Nummer: G626

Art: Warnung

Meldung: line

Bedeutung: Die Anwendung des Parameters `DIRECTINITIAL REDEFINE` auf Strukturkomponenten, welche durch andere Komponenten überlagert werden, kann zu fehlerhaften Initialisierungen auf C-Ebene führen.

Nummer: G741

Art: Warnung

Meldung: Feld `<name>` ignoriert (direkte Strukturinitialisierung)

Bedeutung: Das im TAL-Programm zur Strukturinitialisierung verwendete Feld mit Namen `<name>` wird nicht im C/C++ Code generiert.

Nummer: P001

Art: Fehler

Meldung: Filename erwartet

Bedeutung: Der Translator erwartet den Namen des zu konvertierenden Files.

Nummer: P002

Art: Fehler

Meldung: Filename `%s` nicht lesbar

Bedeutung: Das zur Konvertierung vorgesehene File kann nicht gelesen werden.

Nummer: P003

Art: Warnung

Meldung: Makro `%s` redefiniert

Bedeutung: Innerhalb eines Programmes wurde das genannte Makro redefiniert.

Nummer: S001

Art: Fehler

Meldung: Zu viele offene Files

Bedeutung: Es können systemseitig keine weiteren Files geöffnet werden.

Nummer: S002

Art: Fehler

Meldung: Setupfile `%s` nicht lesbar

Bedeutung: Das File `tal.stp` bzw. das durch die Option `-setup` angegebene File ist nicht lesbar. Unter Umständen ist die Umgebungsvariable `TAL_SETUP` nicht korrekt gesetzt.

Nummer: S003
Art: Fehler
Meldung: Setupfile bereits gelesen
Bedeutung: Der Translator wurde mit mehr als einem `tal.stp` gestartet.

Nummer: S004
Art: Fehler
Meldung: Fehler im Setupfile
Bedeutung: Das File `tal.stp` enthält syntaktische Fehler, eine Weiterübersetzung der TAL-Quelle kann nicht erfolgen.

Nummer: S005
Art: Fehler
Meldung: Variable `TAL_SETUP` nicht lesbar
Bedeutung: Die Umgebungsvariable `TAL_SETUP` kann nicht gelesen werden, die Konvertierung wird abgebrochen, wenn der Konvertierungslauf nicht mit der Option `-setup` gestartet wurde.

Nummer: V001
Art: Fehler
Meldung: `INVOKE`-Indexfile `%s` nicht lesbar
Bedeutung: Das mit der Option `-ivk` angegebene Indexfile kann nicht gelesen werden.

Nummer: V002
Art: Fehler
Meldung: `INVOKE`-Structfile `%s` nicht lesbar
Bedeutung: Das mit der Option `-ivk` angegebene Strukturfile kann nicht gelesen werden.

Nummer: V003
Art: Fehler
Meldung: `INVOKE %s AS %s` nicht auffindbar
Bedeutung: Im TAL-Programm treten `INVOKE`-Direktiven auf, für die im Index- bzw. Strukturfile (Option `-ivk`) keine Beschreibung existiert.

Nummer: V003
Art: Fehler
Meldung: Datenbank fuer `INVOKE %s AS %s` fehlt
Bedeutung: Im TAL-Programm existieren `INVOKE`-Direktiven, ohne daß die `-ivk`-Option gesetzt wurde.

Nummer: Z003
Art: Fehler
Meldung: `THEN`-Zweig fehlt
Bedeutung: Die `IF`-Anweisung besitzt keinen `THEN`-Zweig.

Nummer: Z004
Art: Fehler
Meldung: `ELSE`-Zweig fehlt
Bedeutung: Die `IF`-Anweisung besitzt keinen `ELSE`-Zweig.

Nummer: Z005

Art: Fehler

Meldung: Mehr als ein Ausdruck bei OTHERWISE

Bedeutung: An dieser Stelle darf nach dem OTHERWISE maximal ein Ausdruck auftreten.

Nummer: Z006

Art: Fehler

Meldung: Ausdruck nach OTHERWISE fehlt

Bedeutung: An dieser Stelle muß nach dem OTHERWISE ein Ausdruck stehen.

Nummer: Z007

Art: Fehler

Meldung: Ausdruck wird erwartet

Bedeutung: An dieser Stelle wird ein Ausdruck erwartet.

Nummer: Z008

Art: Fehler

Meldung: Anweisung wird erwartet

Bedeutung: An dieser Stelle wird eine Anweisung erwartet.

Nummer: Z009

Art: Fehler

Meldung: Anweisung wird nach OTHERWISE erwartet

Bedeutung: An dieser Stelle wird eine Anweisung nach dem OTHERWISE erwartet.

Nummer: Z010

Art: Fehler

Meldung: Fehler in Case-Alternative: Label oder Anweisung wird erwartet

Bedeutung: An dieser Stelle eines CASE-Zweiges wird eine Anweisung oder eine Marke erwartet.

Nummer: Z011

Art: Warnung

Meldung: Literal %s mehrfach definiert

Bedeutung: Das genannte Literal wurde redefiniert.

Nummer: Z012

Art: Warnung

Meldung: Define %s mehrfach definiert

Bedeutung: Das genannte Makro wurde redefiniert.

Nummer: Z013

Art: Fehler

Meldung: Fehler in der Konstanten %s

Bedeutung: Der Aufbau der Konstanten entspricht nicht den Konventionen zur Beschreibung von Konstanten lt. TAL-Reference-Manual.

Nummer: Z014

Art: Fehler

Meldung: Variable %s mehrfach definiert

Bedeutung: Die genannte Variable wurde redefiniert.

Nummer: Z015

Art: Fehler

Meldung: ueberlagerte Variable %s undefiniert

Bedeutung: Die an dieser Stelle als zu ueberlagernde Variable wurde noch nicht definiert.

Nummer: Z016

Art: Fehler

Meldung: ueberlagerter Bezeichner %s ist keine Variable

Bedeutung: Das zu ueberlagernde Datenobjekt ist keine Variable.

Nummer: Z017

Art: Fehler

Meldung: Template %s ist nicht definiert

Bedeutung: Der in der Strukturdefinition benutzte Templatename wurde noch nicht definiert.

Nummer: Z018

Art: Fehler

Meldung: %s ist kein Template-Name

Bedeutung: Der in der Strukturdefinition benutzte Templatename steht nicht für ein Template.

Nummer: Z020

Art: Fehler

Meldung: Interner Fehler beim Strukturaufbau

Nummer: Z021

Art: Fehler

Meldung: Struktur %s mehrfach definiert

Bedeutung: Die genannte Struktur wurde redefiniert.

Nummer: Z022

Art: Fehler

Meldung: %s - undefinierter Parameter

Bedeutung: Der genannte Parameter einer Prozedur wurde nicht definiert.

Nummer: Z023

Art: Fehler

Meldung: Template %s darf kein Parameter sein

Bedeutung: Namen von Templates dürfen nicht als Parameter von Prozeduren auftreten.

Nummer: Z025

Art: Fehler

Meldung: Parameter %s mehrfach deklariert

Bedeutung: Der genannte Parameter wurde in einer Parameterliste redefiniert.

Nummer: Z026

Art: Fehler

Meldung: Prozedur/Subprozedur %s mehrfach deklariert

Bedeutung: Die genannte Prozedur oder Subprozedur wurde redefiniert.

Nummer: Z027

Art: Fehler

Meldung: Label %s ist nicht definiert

Bedeutung: Die in einer Sprunganweisung auftretende Marke ist nicht definiert.

Nummer: Z028

Art: Fehler

Meldung: Funktion %s ist nicht definiert

Bedeutung: Die gerufene Funktion ist nicht definiert.

Nummer: Z029

Art: Fehler

Meldung: Literal %s ist nicht definiert

Bedeutung: Das genannte Literal ist nicht definiert.

Nummer: Z030

Art: Fehler

Meldung: Bezeichner %s ist nicht definiert

Bedeutung: Der genannte Bezeichner ist nicht definiert.

Nummer: Z031

Art: Fehler

Meldung: Bezeichner %s muss eine Struktur sein

Bedeutung: An dieser Stelle wird ein Bezeichner erwartet, der eine Struktur beschreibt.

Nummer: Z032

Art: Fehler

Meldung: Entry %s bereits definiert

Bedeutung: Das an dieser Stelle als `ENTRY` definierte Datenobjekt wurde redefiniert.

Nummer: Z033

Art: Warnung

Meldung: %s definiert, aber nicht benutzt

Bedeutung: Das genannte Datenobjekt wurde zwar definiert, bleibt aber in diesem TAL-Programm ungenutzt.

Nummer: Z034

Art: Warnung

Meldung: %s definiert, aber nicht gerufen

Bedeutung: Die genannte Prozedur wurde zwar definiert, wird aber in diesem TAL-Programm nicht gerufen.

Art: Warnung !

Meldung: Bit-Extraction-Operator verwendet

Bedeutung: An dieser Stelle werden Bitextraktionsoperatoren benutzt. Diese Warnung wird nur im Zusammenhang mit der Option `-bit` generiert.

Nummer: Z036

Nummer: Z037

Art: Fehler

Meldung: Parameter %s in Prozedur %s wurde kein Typ zugeordnet

Bedeutung: Dem genannten Parameter der Prozedur wurde kein Typ zugeordnet.

Nummer: Z038

Art: Fehler

Meldung: Filler ausserhalb einer Struktur verwendet

Bedeutung: Eine FILLER-Definition wurde außerhalb einer Struktur benutzt.

Nummer: Z039

Art: Fehler

Meldung: %s muss eine Funktion sein

Bedeutung: Der genannte Bezeichner muß an dieser Stelle eine Funktionsprozedur sein, also einen Rückkehrwert liefern.

Nummer: Z040

Art: Fehler

Meldung: zuviele Argumente zur Funktion %s

Bedeutung: Die genannte Funktion wurde mit zu vielen Parametern gerufen.

Nummer: Z041

Art: Warnung

Meldung: %s besitzt das Attribut VARIABLE nicht

Bedeutung: Die gerufene Funktion besitzt das Attribut VARIABLE nicht, so daß eine Nichtangabe von Parametern nicht erlaubt ist.

Nummer: Z042

Art: Fehler

Meldung: Entry %s darf nicht an dieser Stelle stehen

Bedeutung: An dieser Stelle darf keine ENTRY-Nutzung stehen.

Nummer: Z042

Art: Fehler

Meldung: interner Fehler: Entry %s nicht gefunden

Nummer: Z043

Art: Warnung

Meldung: interner Fehler: fehlerhafte Benutzung von isType

Nummer: Z048

Art: Fehler

Meldung: UNSIGNED Konstanten gibt es nicht

Bedeutung: UNSIGNED-Konstanten werden bisher nicht behandelt.

Nummer: Z049

Art: Fehler

Meldung: Groessenberechnung fuer Variablen dieses Typs ist nicht definiert

Bedeutung: Für Variablen dieses Typs kann bisher keine Berechnung der Speichergröße vorgenommen werden.

Nummer: Z050

Art: Fehler

Meldung: Typfehler: %s muss vom Typ %s sein

Bedeutung: Die an dieser Stelle erwartete Variable muß vom angegebenen Typ sein.

Nummer: Z051

Art: Fehler

Meldung: %s muss ein konstanter Ausdruck sein

Bedeutung: An dieser Stelle wird ein Konstantausdruck erwartet.

Nummer: Z052

Art: Fehler

Meldung: interner Fehler: falscher Typ in Variablenliste

Nummer: Z053

Art: Fehler

Meldung: interner Fehler: falsche Strukturkomponente

Nummer: Z054

Art: Fehler

Meldung: interner Fehler: hier muss ein skalarer Typ stehen

Nummer: Z055

Art: Fehler

Meldung: interner Fehler: Anzahl formaler Parameter falsch

Nummer: Z056

Art: Fehler

Meldung: Standardfunktion %s kann nicht ausgewertet werden

Bedeutung: Die hier benutzte Standardfunktion ist nicht implementiert.

Nummer: Z057

Art: Fehler

Meldung: Bezeichner %s (%s) kann nicht in Ausdruecken benutzt werden

Bedeutung: In den vorliegenden Ausdrücken ist die Verwendung des genannten Bezeichners nicht erlaubt.

Nummer: Z058

Art: Fehler

Meldung: interner Fehler: %s darf nicht in Ausdruecken auftreten

Nummer: Z059

Art: Fehler

Meldung: %s muss ein INT(16)-Ausdruck sein

Bedeutung: Das genannte Datenobjekt muß ein INT-Ausdruck sein (16 Bit).

Nummer: Z060

Art: Fehler

Meldung: in Initialisierungen dürfen nur einfache Typen (keine Felder, Zeiger oder Strukturen) auftreten

Bedeutung: Initialisierungen von Feldern müssen in jedem Fall berechenbare skalare Ausdrücke besitzen. Die Benutzung von Struktur- und Feldnamen ist nur dann zulässig, wenn sie in Zusammenhang mit einer Operation erfolgen, die einen Konstantwert liefert, wie beispielsweise die `$LEN`-Funktion.

Nummer: Z061

Art: Fehler

Meldung: Elemente einer Konstantenliste müssen den gleichen Typ haben

Bedeutung: Alle Elemente der Konstantenliste müssen vom gleichen Typ sein.

Nummer: Z062

Art: Fehler

Meldung: Typkonflikt bei Operator `%s` (`%d`. Argument)

Bedeutung: Für den genannten Operator kann keine Berechnung vorgenommen werden, da die benutzten Operanden keine kompatiblen Typen besitzen.

Nummer: Z063

Art: Fehler

Meldung: count-unit muss `BYTES`, `WORDS` oder `ELEMENTS` sein - oder leer

Bedeutung: Als Zählinheit in einem `MOVE`-Statement wurden andere als die genannten Bezeichner benutzt.

Nummer: Z064

Art: Fehler

Meldung: links und rechts von einer Zuweisung müssen kompatible Typen stehen

Bedeutung: Eine Wertzuweisung kann nur ausgeführt werden, wenn beide Seiten kompatible Typen besitzen.

Nummer: Z065

Art: Fehler

Meldung: `%s` muss zwischen `%d` und `%d` liegen

Bedeutung: Ausdrücke hinter dem Schlüsselwort `FILLER` bzw. in Bitextraktionsoperationen dürfen einen bestimmten Wertebereich nicht über- bzw. unterschreiten.

Nummer: Z066

Art: Fehler

Meldung: Typfehler bei Funktion `%s`, `%d`. Argument

Bedeutung: Der aktuelle Parameter einer Funktion besitzt einen Typ, der nicht mit dem des formalen Parameters korrespondiert.

Nummer: Z067

Art: Fehler

Meldung: Initialisierung entspricht nicht dem Variablentyp

Bedeutung: Der Typ des Initialisierungsausdrucks ist inkompatibel zur Variablen, die initialisiert werden soll.

Nummer: Z068

Art: Fehler

Meldung: `%s` muss eine Variable sein

Bedeutung: Der genannte Bezeichner muß eine Variable sein.

Nummer: Z069

Art: Fehler

Meldung: Typkonflikt bei RETURN-Statement

Bedeutung: Der Typ des Ausdrucks im RETURN-Statement ist inkompatibel zum Rückkehrtyp der Funktion.

Nummer: Z070

Art: Fehler

Meldung: Hier darf kein RETURN stehen

Bedeutung: Eine RETURN-Anweisung ist an dieser Stelle unzulässig.

Nummer: Z071

Art: Fehler

Meldung: MAIN-Prozedur kann keine Funktion sein

Bedeutung: Die mit dem Attribut MAIN gekennzeichnete Funktion darf keinen Rückkehrtyp enthalten.

Nummer: Z072

Art: Fehler

Meldung: MAIN-Prozedur darf keine Parameter haben

Bedeutung: Die mit dem Attribut MAIN gekennzeichnete Funktion darf keine formalen Parameter besitzen.

Nummer: Z073

Art: Fehler

Meldung: Indirekte Variablen in Subprozeduren nicht erlaubt.

Nummer: Z076

Art: Fehler

Meldung: FORWARD-Deklaration hat verschiedene Parameterliste

Bedeutung: Die in einer FORWARD-Deklaration angegebene Parameterliste unterscheidet sich von der in anderen FORWARD-Deklarationen der gleichen Funktion bzw. von denen der Funktionsimplementierung.

Nummer: Z077

Art: Fehler

Meldung: Konstantenauswertung %s im Translator vorerst nur fuer INT-Ausdruecke implementiert

Bedeutung: Eine Auswertung der genannten Konstante kann in der vorliegenden Translatorversion nicht erfolgen. An dieser Stelle können nur INT-Ausdrücke berechnet werden (16 Bit).

Nummer: Z078

Art: Fehler

Meldung: Konstantenauswertung fuer %s im Translator vorerst nicht implementiert

Bedeutung: Eine Auswertung der genannten Konstante kann in der vorliegenden Translatorversion nicht erfolgen.

Nummer: Z079

Art: Fehler

Meldung: einer %s-Variablen kann keine String-Konstante der Laenge %d zugewiesen werden

Bedeutung: Der genannten Variablen kann keine Zeichenkettenkonstante der genannten Länge zugewiesen werden.

Nummer: Z080

Art: Fehler

Meldung: Strukturlayout zu %s fehlt

Bedeutung: Zu dieser Struktur wurde noch kein Layout definiert, obwohl es an dieser Stelle benötigt wird.

Nummer: Z081

Art: Fehler

Meldung: Sorry, Code-Generierung fuer %s noch nicht implementiert

Bedeutung: Eine Codeerzeugung für den genannten Ausdruck ist in dieser Version nicht realisiert.

Nummer: Z083

Art: Warnung

Meldung: %s kann zu Laufzeitfehlern fuehren

Bedeutung: Mit dieser Warnung wird eine Situation beschrieben, die aufgrund von Inkompatibilitäten zwischen TAL und C/C++ zu Laufzeitfehlern nach der Konvertierung führen kann.

Nummer: Z084

Art: Fehler

Meldung: interner Fehler: keine Konvertierung von %s nach %s definiert

Nummer: Z085

Art: Fehler

Meldung: %s muss ein Parameter sein

Bedeutung: Der genannte Bezeichner muß ein Parameter sein.

Nummer: Z086

Art:

Meldung: Uebersetzung von %s: %d Fehler, %d Warnung(en) und %d sorrys

Bedeutung: Die Konvertierung des TAL-Programmes ist beendet, es traten die genannte Zahl von Fehlern, Warnungen und Sorrys auf.

Nummer: Z087

Art: Fehler

Meldung: Interner Fehler: falsche Groesse (%d) fuer read-only-feld berechnet

Nummer: Z088

Art: Fehler

Meldung: Die Lizenz fuer *cgn* ist abgelaufen

Nummer: Z089

Art: Fehler

Meldung: *cgn* ist fuer diesen Rechner nicht lizenziert

Nummer: Z090

Art: Fehler

Meldung: Interner Fehler: Grammatik (Zeile %d), %s

Nummer: Z091

Art: Warnung

Meldung: globale Deklaration besteht aus mehreren Sections
Bedeutung: (Eine einzige globale Deklaration setzt sich aus mehreren Sectionen zusammen.)

Nummer: Z092
Art: Mitteilung
Meldung: File: %s, Section: %s (Zeile %d)
Bedeutung: (Gibt die Files und Sections des Quelltextes aus.)

Nummer: Z093
Art: Fehler
Meldung: interner Fehler: fehlerhafter C-Prototyp (

Nummer: Z094
Art: Fehler
Meldung: %s C-Prototyp in setup-File mit anderer Parameteranzahl definiert

Nummer: Z095
Art: Fehler
Meldung: interner Fehler: Template

Nummer: Z096
Art: Warnung
Meldung: Differenz von `STRING`- und `INT` Zeigern fuehrt zu Laufzeitfehlern

Nummer: Z097
Art: Warnung
Meldung: Summe von `STRING`- und `INT` Zeigern fuehrt zu Laufzeitfehlern

Nummer: Z098
Art: Fehler
Meldung: Parameter %s darf keine direkte Struktur sein

Nummer: Z098
Art: Warnung
Meldung: %s: Binaere `FIXED`-Konstanten werden nicht unterstuetzt

Nummer: Z099
Art: Fehler
Meldung: %s: Adressen von Marken werden in C nicht unterstuetzt

Nummer: Z298
Art: Warnung
Meldung: %s-Parameter der Laenge %d wird abgeschnitten
Bedeutung: An einen `STRING`-Parameter der Länge 1 wird eine Zeichenkette der Länge 2 übergeben.

Anhang C: Literaturverzeichnis

Literatur:

[1] Schmidt, T.: Benutzerdokumentation Zerteiler.
Technischer Report, pro et con GmbH, Chemnitz, 2006

[2] Erdmenger, U.; Loos, A.; Schmidt, T.: Abbildungsdokumentation TAL To C/C++.
Technischer Report, pro et con GmbH, Chemnitz, 2006