Pmm

Michaela Prüß

COLLABORATORS

	TITLE : Pmm	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Michaela Prüß	April 14, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Pmm

1.1 Pmm - Inhalt

PMM.Guide Version 1.3 11.04.1999

Inhalt dieser Guide:

Copyright Pmm, Pmm-Tools / Adresse Nutzungslizenz Software und Rechte dritter

Wie und warum, noch ein Make-Tool? Allgemeine Übersicht zu Pmm Planung, wie es weiter geht Einige generelle Dinge

Pmm - Bedienung, Parameter Das Hauptprogramm PmmCmp - Übersicht Der Makeersatz PmmLibr - Funktion und Bedienung Ein Libr-Programm PmmScan - Funktion und Bedienung Der Depend-Scanner PmmTool - Funktion und Bedienung Eine Funktionssammlung PmmView - Übersicht und Probleme Das Anzeigeprogramm GSort - Bedienung, Lizenz Ein Sort-Befehl

Hinweise zur gateway.library

Eine ToDo-Liste, nicht komplett Bekannte Fehler Schlußwort

1.2 Warum

"Projekt Make Manager"

Warum? - Weil mir die derzeitigen Alternativen nicht gefallen: + Make: Zu unflexibel oder es kommen so gigantische makefiles dabei raus, das man die nicht mehr handhaben kann. Das führt irgendwann mit Sicherheit zu Fehlern. + Depend-Scanner: Passen nie zu dem Make, mit dem man arbeitet, der Output ist selten wirklich brauchbar. + Make-Oberflächen: Alle, die ich kenne, sind entweder komplizierter als die Programme, die man damit am Ende schreiben will oder funktionieren einfach nicht. Und wenn sie funktionieren, dann garantiert nur mit einem Compiler, den man selber nicht benutzt. + Ich selber habe den alten Aztec 5.2a gehabt und bis vor kurzen damit gearbeitet. Und bevor ich mich am Ende im Freeware-Bereich für vbcc entschieden habe sind SAS & Storm auf der kommerziellen Seite, Dice und GCC/GNU geprüft worden. Zu SAS hatte ich nie den Draht und Storm ist so, ANDERS und FREMD, also ich weiß in der Regel, was wie und wo passiert und wie ich ein Problem aus der Welt schaffe, Storm ist mir dafür zu eigen. DICE, nun, es fehlt(e?) die PPC-Seite. Da hätte ich Aztec behalten können. GNU, tja, etwas zu viel des guten, und wen kann man mal fragen, wenn was brennt? Irgendeine UseNet-Gruppe? Na ja, ist ja auch letztendlich nicht entscheident. Viel wichtiger ist, ich kann mit PMM jeden der genannten Compiler (außer vermutlich Storm) benutzen. Ich müßte nur die Regeln in Make/Pmm.Cfg umsetzen. Die maximale Stufe der an der Erzeugung eines Objektes beteiligter Programmschritte könnte das einzige Problem sein. Das Maximum habe ich an vbcc angepasst: 1.) source -> object: .c -> .asm -> .s -> .o 2.) objectlist -> endfile: alle .o -> Kommando Also (1.) Preprozessor, Compiler, Scheduler und Assembler, solange bis die Liste erledigt ist und dann (2.) einmalig die Übergabe der Liste an ein Kommando. Das ist in der Regel ein Linker, aber wie im Demofile als Beispiel aus meinem Projekt zu sehen kann es auch ein Libr-Kommando zum Erstellen von .Lib-Dateien sein. Hier wird das $L=, das sonst das Linkkommando enthält durch ein 'List #?.o LFORMAT...' ersetzt, dann folgt das Execute des per List erzeugten Batches. Dies steht in einer Zeile die sonst als Pre-Kommando nach Link benutzt werden kann. Jede Stufe kann einfach ausgelassen werden, indem man eines der Symbole (.c,.a,.s,.o) nicht benutzt. Mehr wären möglich, sind aber nicht

implementiert. Die Gruppe der .o-Files geht zum Schluß gesammelt als letzter Schritt in den Linker zum Endprodukt. Hinweis: Außer mit vbcc wird derzeit das Fehler- protokoll nicht laufen. Im Moment ist diese Funktion allgemein nicht fertig. Später wird es eine Liste zur Erkennung geben die angepasst werden kann.

1.3 Was kann es?

"Was kann PMM zur Zeit?"

- Multiples Make: gleiche Sourcen und Include-Strukturen werden
auf gesplitteten Wegen zu Programmen für die
unterschiedlichen CPU-Varianten. Es kann auf
vielerlei Wege und sehr differenziert angegeben
werden, was sich im Einzelfall ändert.
- Zentrales Make: Ein 'PMM-Makescript', ein Strukturverzeichnis,
eine Projektdatei und eine Projektliste
genügen um beliebig viele Programme mit bis zu
10 CPU-Typen, 10 Compilern, 10 Linkern, 10
unterschiedlichen Outputs und/oder 10 Ziel-
rechnern zu verwalten.
- Eigener Depend: Der interne Dependscan (suche nach Include-
files) kann einen neuen Scan durchführen,
wann immer das nötig erscheint und er macht
alles weitere selbst, wenn einmal das Programm
im Projekt eingerichtet wurde.
- Sub-Funktionen: Sub-Funktionen (Library's, Programme, .Lib's
usw.) werden mit verwaltet. Stellt sich beim
Progammieren im Bereich a, Programm aa heraus,
das die Lib XYZ gebraucht wird und diese ist
nicht übersetzt, dann startet man dort, wo man
gerade steht, PMM mit nur zwei Parametern. Das
Sub-Modul wird übersetzt und man steht immer-
noch an der gleichen Stelle, wie vorher.
- WorldWide-Make: Überall auf dem Rechner PMM starten, angeben,
was hergestellt werden soll, und fertig. Den
Rest macht das System alleine.
- Kommandostarts: Klar, kann jedes Make. Nur, wenn dort ein DOS-
kommando scheitert, dann bricht das ganze System
ab. PMM ignoriert alles, außer die Fehler werden
vom Compiler-System ausgelöst.
- 10 auf einen Streich:

Was kleine, tapfere Schneider konnten, das sollte dieses System wohl auch hinbekommen. Wird PMM mit -A gestartet, dann werden alle Versionen des gerade eingestellten Programmes hergestellt. Und über A steht auf der Tastatur das Q wie Qualität und Quantität. PMM Q ist die Methode dem System zu sagen das man mehr will als alles, nämlich wirklich alles und das Q-UICKY. Q schaut nicht nach, ob und welche Dateien zu neu sind, es rennt nur einmal durch die Projektliste und prüft, ob alle .o-Files und alle Endprogramme/Lib's usw. vorhanden sind. Alles was fehlt wird hergestell. Q-uick meint, das die DEPEND-Listen (.h-Files) nicht geprüft werden. Entsprechend schneller wird ein fast komplettes Projekt erstmal ergänzt. Mit A wird schließlich alles hergestellt unter Berücksichtigung aller Prüfungen. Ob Q oder A benutzt werden bleibt an sich jedem selber überlassen. Hat man aber z.B. alle Objekt-Files gelöscht und will alles neu übersetzen reicht Q aus. Auch nach einem Abbruch und der Korrektur eines einzelnen .c-Files kann mit Q wieder weitergemacht werden. Sobald aber .h-Files angefasst wurden ist davon abzuraten.

- make the makefile bevor make:

Mit I (alles) oder -i (Einzelprogramm) es es schließlich sogar möglich vor dem Make die Dependliste(n) neu zu generieren. Wer hat nicht schon mal eine umfangreiche Veränderung vorgenommen und dabei in einem (oder gar mehreren) C-Files ein neues Include-File eingebaut und vergessen es an irgendeiner Stelle nachzutragen? Wer einen Dependscan automatisch integriert laufen hat, der hat sicher keine Probleme. Wer das nicht hat, bei dem kann man drauf warten, das irgendwann ein Fehler entsteht weil Make nicht mitbekommt, das irgendwas neu übersetzt werden muß.

Das berühmte -CLEAN- im Makefile zum Löschen aller Objekt-Files zeigt ansich, wie sehr man dem make doch nicht traut. Wozu make, wenn man immer alles

neu übersetzt?

- weitere Tools: Gibt es, siehe Inhalt. Die Tools werden weiterhin

ausgebaut. Dabei werden je nach Bedarf neue Programme

oder Funktionen entstehen. Das Endziel ist, das die

Batches vom Programm 'geschluckt' werden un sich am

Ende alles von einem zentralen Punkt steuern läßt.

1.4 Wie es weiter geht

"Planung: Was kommt als nächstes?"

- Intui ohne MUI: Eine Intui-Oberfläche ist bereits in Arbeit. Sie ist eine Eigenkreation, wie das ganze System. Keine MUI-Oberfläche. Sicher nicht so eindrucksvoll, aber dafür kostenlos und speicherschonend. Ich habe nichts gegen Shareware, ganz besonders dann nicht, wenn solch hoher Aufwand, wie z.B. in MUI, getrieben wurde. Und das schöne an Shareware in der Grundidee ist ja, das man sich alles in Ruhe ansehen kann und das bezahlt, was man wirklich nutzt und das einem gefällt. Mit MUI ist eine Situation entstanden die vollkommen aus der Art schlägt. Ein Meisterstück (kommerziell betrachtet!!!) vom Autor und eine, wie ich finde, unverfrorene Frechheit einiger Software- anbieter. Da soll man also für ein kommerzielles Programm wie IBrowse, AWeb, Voyager und andere die Software bezahlen und danach, um sie 100% und nicht nur 80% nutzen zu können die Sharewaregebühr an MUI? Das dürfte bis heute einmalig sein. Wenn schon keine Voll-Lizenz für MUI dem kommerziellen Programm bege- legt wird, dann doch wenigstens eine, die alle MUI- einstellungen für eben dieses Programm öffnet. Was bei dem exestierenden Aufbau von MUI sicherlich zu machen wäre. Oder, siehe Miami, man schafft eine Alternative die wahlweise den Betrieb ohne MUI er- möglicht. Wenn ein Freeware-Programm wie Miami ein Sharesystem wie MUI benutzt ist das was anderes. Aber die kommerziellen Systeme sollten mal nachdenken was sie da tun. Und da ich persönlich diese Zustände nicht unterstütze, mache ich dann halt lieber meine Dinge selber und bleibe beim guten, alten Intuition.

- Versionsverwaltung(en): Ist auch schon begonnen. Zur Zeit nutze ich noch HVC, niedliches Tool und normalerweise ausreichend. Aber ich brauche ganz persönlich eine Versions-Stufen- Verwaltung: Projekt: VERS / REV / PATCH MODUL, Bereich im Projekt, ebenfallls mit V/R/P

- Ende BETA-Status: Zur Zeit ist das ganze eine BETA-Version. Zum Teil mit nervigen Einschränkungen und Problemen. Dieses soll beendet werden, sobald sicher keine Fehler mehr auftreten und zumindest eine kleine Oberfläche vorhanden ist.

- Ausbau der zum Teil schon begonnenen Source-Code-Pflege, Update- und Backup-Funktionen. Von der TAB-Funktion zum CRef über einige Prüfläufe soll das Programm alles können und sinnvoll verbinden. Das Ziel ist: Alles in einem Lauf, denn was kann es bequemeres geben?

- Vom Compiler unabhängig(er) arbeiten. Ich bin dabei einige Grund- funktionen meiner BBS-Software mit allgemeinem C-Standard-KnowHow zu verbinden und zu bündeln. Am Ende sollten .Lib-Dateien und shared-libraries entstehen, die mit FD-Dateien und passenden Funktionen kombiniert dem PMM beige- legt werden. Warum mit abweichenden Implementationen der Compiler hadern, wenn man sich das eigentlich ersparen kann? Nur, ich will keine IXEMUL.library haben, die 10 mal größer ist als die Programme die sie nutzen und die enormen Speicher verbraucht, nur weil es UNLINK sein muß, statt DeleteFile. Einige kleine, abgerundete Libs und Library's sollten für gezielten Einsatz besser sein. Die Link-Libs kann man ja sowohl als große Brocken als auch als kleine Module halten (mache ich jetzt schon). Dabei fällt mir ein, ich brauche jemanden, der sich mal ein Stück Assembler-File (Zugriff aus einer Library auf die DOS-Library) ansehen kann, das unter VBCC nicht will. Das ist direkt für Aztec geproggt. Ich bin dabei, eine ganz alte Tante vom Aztec auf den vbcc zu bringen, die inzwischen fast vergessene IntuiSub.Library. Bei Aztec->vb habe ich ja schon meine Erfahrungen gemacht.

1.5 Generelles

"Generelles zum Projekt Make Manager"

Ich werde Zusatzfunktionen und Service-Funktionen nur zum Teil in die jetzigen Programme einbauen. Einiges wird, weil dann besser einzeln nutzbar, als Zusatz-Programme realisiert. Diese können dann auf Wunsch auch ersetzt werden. Eniges wird sogar in Shared-Library's aufgenommen, was das ganze dann direkt einsetzbar macht.

Sofern ich nicht am Ende alleine und als einzige mit dem System mei- nen Kram mache, sondern einige Leute mehr wird sich sowieso alles "entwicklen".

1.6 Nutzungslizenz

"Projekt Make Manager Lizenz"

---------- Freeware, Public Domain, Shareware, AFT, GNU-Public Lizence, Kommerz ---------- Der Verwendungszweck der Software, die hiermit erstellt wird, kann durchaus in jede dieser Gruppen passen. Ob es nun ein 10 Minuten Tool ist oder ein Giga-Projekt das teuer vermarktet wird, zunächst einmal ist dieses Tool kostenlos. Wer's wirklich einsetzt, der sollte aber mal was von sich hören lassen, damit ich weiß, ob es sich lohnt, mehr als meinen Kram zu machen. Wer eine Non-Free-Software herstellt (Share oder Kommerz), sollte mal drüber nachdenken, das auch Freeware Zeit und Nerven kostet und genauso viel Arbeit macht, wie sein Produkt. Ein Keyfile bzw. eine Nutzungslizenz kosten andererseits auch nichts und sind jederzeit eine gern gesehene Anerkennung. Wer's nicht tut, nun, dem mache ich auch keinen Vorwurf, das muß jeder selber wissen. ---------- Freeware, Public Domain, Shareware, AFT, GNU-Public Lizence, Kommerz ----------

1.7 Pmm

"Pmm (Aufruf und Parameter)"

Pmm "merkt" sich zunächst mal, was der Benutzer tut. Das heißt, wenn man ein Programm bearbeitet und das mit Parametern aufgerufen hat, dann genügt es beim nächsten Anlauf nur Pmm zu starten, die Parameter werden wieder gesetzt. Die hiervon betroffenen Parameter werden unten speziell gekennzeichnet.

Diese Funktionen bearbeiten die Projektliste (alle Programme): ALL bzw. OPT gilt jeweils für jedes Programm:

Pmm Q - Quick and Dirty, alle fehlenden Objekte und Exec's herstellen, nur .o gegen .c prüfen, die .h-Files werden nicht geprüft. Pmm q OPT - Alternativ zu Q, wenn nicht ALL gesetzt werden soll, sondern statt dessen OPT. Pmm A - All, herstellen aller Einzelobjekte mit dem Para- meter ALL. Pmm a OPT - Alternativ zu A, wenn nicht ALL gesetzt werden soll, sondern statt dessen OPT. Pmm I - Wie A, nur zuvor wird für jedes Progamm ein neuer Scan aller .c-Files und der Includes durchgeführt. Pmm i OPT - Alternativ zu I, wenn nicht ALL gesetzt werden soll, sondern statt dessen OPT.

Diese Funktionen beziehen sich auf einzelne Programme. Wobei ein einzelnes Programm sich in die CPU-Versionen aufteilt.

Pmm ? - Informationen über das Programm. Pmm -?, -h, -H - Kurze Hilfe anzeigen. Wirkt zugleich wie -n! Pmm -a - Aktuelles Programm mit Option ALL übersetzen. Ist die Kurzform von '-o ALL'. Pmm -n - Nicht übrsetzen (NO MAKE). Schaltet bei -c/-i den anschließenden MAKE aus. * Pmm -q [0-9] - Quiet-Modus (Default=8). Je höher der Modus, desto weniger Ausgaben werden produziert. 9 schaltet jede Anzeige aus, 8 ist der Benutzermodus mit wenig aber ausreichenden Anzeigen. 0-7 sind immer gesprächiger werdende Anzeigen (0 ist der DEBUG-Modus). Da diese Ausgaben nur zu Testzwecken dienen, sind sie nicht unbedingt schön. Im Moment ist die Ausgabe via PmmView bereits in Arbeit und wird an vielen Stellen als Standard geschaltet. -q (0-9) schaltet die Anzeige via PmmView ab! Bzw. mit der Variation -q 88 kann PmmView aktiviert werden. Die Funktionen (Q,q,A,a,I und i setzen -q 88 auto- matisch). Pmm -E - Ruft den Requester zum Einstellen des Default-Editors auf. Die Option startet sofort, die Überprüfung der Parameter wird solange unterbrochen. Pmm -c - Depend-Liste neu erstellen. Pmm -i - Depend-Liste neu scannen. Pmm -C - Projektconfig im Editor (siehe -E) anzeigen. Auch dieser Parameter startet sofort. * Pmm -p [prog] - Aktuelles Programm auf 'prog' wechseln. * Pmm -o [obj] - Objektliste aus dem Projektfile (z.B. ALL) zur übersetzung benutzen.

Die mit * markierten Einstellungen werden gespeichert. Wird zum Beispiel PMM -p MyProg -o ALL -q 5 eingegeben, dann bewirkt ab sofort der Aufruf ohne Parameter genau das gleiche. Wird Später Pmm -o 040 aufgerufen, dann bleiben die beiden anderen Parameter gestehen und nur die Objektliste ändert sich. Normalerweise sollte es reichen, jeweils -p zu verstellen und den Rest beizubehalten. Soll ein anderes, als das aktuelle Programm mit -c eingelesen werden, dann muß der Schalter -p verwendet werden. Er muß VOR -c stehen. Das gleiche gilt für die optionale Verwendung von -n. Also so Pmm -p MyProg [-n] -c (hier startet das Einlesen!).

Special-Hack's:

Ich brauchte eine Funktion, die sicherlich kein Standard ist. An genau 3 Stellen muß ich in meinen Programmen den Optimizier von -O=3 auf -O=1 umsetzen, weil sonst der Speicher nicht reicht. Für diesen Fall wird bei jedem .c-File geprüft, ob es auch ein .pmm-File gibt. Falls ja wird die Sonderconfig für dieses eine Source-File eingelesen und übersteuert das Projekt. Daduch ergibt sich folgende Hyrachie: Datei 'Pmm.Cfg' Datei 'Projekt' mit :BEGIN * Datei 'Projekt' mit :BEGIN progname Sourcefile:

'sourcefilename.pmm' Es kann in jeder dieser Ebenen gearbeitet werden. Jede Kombination sollte funktionieren. Zu beachten ist, wenn mit % gearbeitet wird, dann muß der Default immer vor den Einzelconfig's stehen.

Da bei mir an zwei Stellen Probleme mit den Prototypen auftreten mußte auch hier eine Lösung her, bis der eigene Scan steht. Wird an einen Filenamen _PMM angehangen, dann wird dieses vom Proto-Typenbatch statt des generierten Scan-Batches ausgeführt. Die Lösung ist mies und auch nicht für länger als unbedingt nötig angedacht. Bitte nicht nachmachen, wie gesagt, das soll nicht so bleiben. Der Proto-Scan ist schon fast fertig, zur Zeit aber noch verwanzt.

1.8 Schlußwort

"Und noch eine kleine Anekdote zum Schluß (oder Murphy's Gesetze Live):"

Ich DACHTE (das war ein Fehler!) ich hätte alles wesentliche bedacht. Und so begann Pmm sich und seine Tools selber zu übersetzen. So entfernte ich dann schließlich irgendwann die makefiles vom make und alles war bestens. Nur, eine kleine Kleinigkeit, die hatte ich übersehen. Und schon passierte es: Pmm ging in einen Makelauf, übersetze sich selber und dann die (damals noch zwei) Zusatzprogramme. Nachdem PmmCmp übersetzt war stieg dieses umge- hend aus. Meldung: Die gateway.library sei zu alt. Pmm lief ab selbigen Moment natürlich auch nicht mehr. Vorher war es nur gelaufen, weil es ja vor der Selbstübersetzung gestartet worden war. Der Fehler war klar, die Library hätte zuerst übersetzt werden müssen. Danach wäre zudem noch ein tausch der geladenen Library nötig geworden. Aber das wäre ja gegangen. Nur, die Library war in diesem Fall noch garnicht übersetzt. Die Programme wollten mit einer Library arbeiten, die es erst nach dem Lauf gegeben hätte. Dumm gelaufen, weil: Kein Pmm, keine Library; keine Library, kein Pmm. Zum Glück hatte ich ein Backup mit den Programmen und nicht, wie ich es sonst meist mache, nur die Quellen. Seit diesem Tag gibt es eine Zusatz- sicherung des Pmm-Systems. Aber sowas mußte einfach passieren. Weil sowas irgendwie einfach immer passiert, jedenfalls mir.

Murphy's Gesetze sind wohl das einzige, worauf man sich garantiert und immer verlassen kann!

1.9 Copyright

Michaela Prüß

Alt-Buckow 45

12349 Berlin

eMail: Michaela@Flagranti.Net

Genehmigung das Archiv auf ihren Servern, Disketten und/oder CD-
Serien zu veröffentlichen. Dies gilt im besonderen auch für die
AmiNet-PD-CD-Serie. Das Erstellen und Verbreiten von guter PD-
Serien (wie halt Fish und AmiNet) ist zeitaufwendig und es ent-
stehen kosten. Solange die Einnahmen den Kosten in etwa entsprechen
darf veröffentlicht werden. Ich sehe hier die AmiNet-Serie als
Maßstab, da sie zwar Gewinne für die Vertreiber abwirft aber diese
im Rahmen hält. Alles, was für eine CD mehr als 5.-DM über AmiNet-
Preis liegt ist in meinen Augen keine PD und somit von der Lizenz
zur Verbreitung ausgeschlossen.
Es werden keine Garantien für das Funktionieren des Programmes
oder evtl. durch das Programm beschädigter Daten übernommen.
Wer das Programm einsetzt, der tut das auf eigenes Risiko und
eigene Verantwortung.

1.10 Fremdsoftware

"Pmm benutzt Software mit rechten dritter!"

"vbcc" in einer Version 0.6 vorhanden sein. Die Versionen Phase-5 PowerUP-CD, AmiNet 0.6a / Frank Wille 0.6c (mit WarpOS-Beta) und der Version 0.6e (Sourcen auf der Homepage von Volker Barthelmann) auf jeden Fall funktionieren. Diese habe ich selber und arbeite damit.

"vbcc Zusatzsoftware" Die Assembler, Linker, Tools usw. die in den Scripten aufgerufen werden, sollten alle beim vbcc-Installtionssatz bei sein. Ausgenommen die WarpOS-Beta (die übrigens BETA nicht verdient hat, so schön wie sie funktioniert). Zu finden bei Frank Wille auf seinen Homepages für pasm / PhxAsm / PhxLnk zusammen mit vielen anderen nützlichen Dingen.

PmmLibr basiert auf den Sourcen zu Libr, ist aber vollkommen über- arbeitet und in vielen Teilen vollkommen neu geschrieben, da mich in aLib die Abfragen stören (ist im integrierten Betrieb einfach nicht zu gebrauchen) und Libr nur in einer 68000-Version vorliegt die nicht mehr so ganz zeitgemäß ist. Hinzu kommt das es (wie aLib) nicht auf PPC läuft und ich das ganze System dorthin portieren will, sobald ich ein Problem mit WarpOS gelöst habe. Mal davon ab- gesehen, das beide Programme Fehler haben und ich außerdem noch ein Zusatzfeature benötigte. Dumm war, die Alib-sourcen hätten nur einige Anpassungen benötigt. Ich hatte sie jedoch übersehen. Als ich schon fast fertig war bin ich drüber gefallen. Pech...

1.11 Bekannte Fehler

"Bekannte Fehler"

Es gibt noch einige offene Punkte, wie die unfertige Fehler- erkennung Linker und Assembler. Echte Fehler sollten in der jetzigen Version keine mehr drin sein.

1.12 GSort

"GSort"

Das beiliegende Programm GSort ist Bestandteil eines anderen Paketes, der Gateway-BBS-Software. Es ist KEINE PD und sein Nutzungrecht beschränkt sich ausschließlich auf die Verwendung als Quicksorter in PMM. Das sonst zum BBS-System Gateway-5 gehörende Tool ist im Originalzustand. Wer auf die dumme Idee kommt damit herumzuspielen, bitte, aber nicht

beklagen wenn es sich eigenwillig verhält. Da das Programm normalerweise zum Sortieren von Dateien im Mailboxeinsatz dient und deren Sortierung zum Teil zuerst einmal eine Umstellung der Daten notwendig macht, kann man im Prinzip alles damit sortieren. Die Mailbox hat dafür intern alle Aufrufe integriert. Bevor sich jemand damit irgendwas zerstört, hier eine Anleitung und die sehr wichtige Warnung: GSort sortiert immer die angegebene Datei. Das heißt: Keine Kopie, keine Sicherung, im Fehlerfall oder bei einem Rechnerabsturz während des Sortierens muß mit dem Verlust der Daten gerechnet werden. Fehler hat es in den letzten Jahren im Sortprogramm keine gegeben, ausgenommen es sind fehlerhafte Parameter zum Aufruf benutzt worden.

Aufruf ist: GSort <dateiname> <satzlaenge> <byte> <laenge> [1] <dateiname> Die Datei, die sortiert werden soll, sie wird nach der Sortierung ohne vorherige Kopie überschrieben! <satzlaenge> Bytes pro Zeile incl. Linefeed. Falsche Angaben können die Datei unbrauchbar machen. Bei variabler Laenge muß das maximum angegeben werden. <byte> Ab welchem Byte (beginnend bei 0) steht das Sortkriterium über das sortiert wird. <laenge> Anzahl Bytes des Sortkriteriums. [1] Option: Brettindexdatei über Datum sortieren. Diese Funktion besser nicht benutzen. Sie ist direkt von der gerade aktuellen BBS-Software abhängig.

1.13 Gateway-Library

"Die gateway.library"

Eigentlich gehört diese Library zu Gateway-5 (wie auch GSort). Aber weil ich sie in Pmm an vielen Stellen einsetze und sie daher ohnehin mit dem Programm ausgeliefert wird, habe ich sie für die externe Nutzung komplett dokumentiert. Ob sie für irgend jemanden überhaupt sinnvoll ist, das hängt vor allem sehr stark von seinem Compiler ab. In erster Linie finden sich hier Stringfunktionen, meist aus dem Unix-Bereich, aber nicht nur.

In vbcc ist keine einzige der integrierten Funktionen enthalten, Nutzer der STORMAMIGA.LIB werden kaum etwas sinnvolles damit anfangen können. GCC hat sicherlich die meisten Funktionen auch, in der ixemul. Aber die ist bekanntlich recht groß und hat recht eng gesetzte Lizenz- bedingungen. Die Doku zur Library sollte sich an gleicher Stelle be- finden wie diese Doku. Falls nicht, dann ist das Archiv nicht mehr im Originalzustand und ich rate dringend dazu sich ein komplettes Original zu besorgen.

1.14 PmmCmp, das Makemodul

"PmmCmp"

PmmCmp ist das eigentliche MAKE-Progamm. Das heißt, es führt das Projekt aus. Gesteuert wird es von Pmm, dem Hauptprogramm. Dieses ruft z.B. bei einer Listenverarbeitung (A/Q/I) das Subprogramm (mehrfach) auf. PmmCmp ist nicht für eine direkte Verwendung gedacht. Es soll an dieser Stelle genügen zu wissen, wozu es gut ist. Seine Aufrufschnittstelle kann sch jederzeit ändern. Die Benutzersteuerung erfolgt deshalb immer und ausschließlich über Pmm.

1.15 Libr-Ersatz

"PmmLibr - Noch ein Libr-Programm"

PmmLibr ist das gleiche wie Libr oder aLib. Zum Teil sind Sourcen des Original-Libr überarbeitet enthalten. Es dient zum Erstellen und Bearbeiten von .Lib-Dateien.

Syntax: PmmLibr -adlrv <Library> [file ...] oder PmmLibr -Fv <Library> Objectlistendatei a A r R -a -A -r -R: Library anlegen und/oder bearbeiten d D -d -D : Files aus Library löschen. l L -l -L : Auflisten der Files in der Library. f F -f -F @ -@ : Statt .o-Files eine Datei mit der Liste aller Objekt-Files (F und @ sind an PhxLnk und vlink angelehnt). Es werden typische Linker-Listen 'p1.o p2.o p3.o ...' genauso akzeptiert wie durch Newline getrennte Listen. v V : verbose, sehr viele Informationen und Ausgaben (Fehlersuche, Debug).

Groß-/Kleinschrift der Optionen ist unerheblich. Grund für diese neue Version war, das aLib störende Abfragen hat und das alte Libr- Programm nur für 68000 vorliegt. Beide Programme haben Fehler und Macken bis zum GURU! Wer's sehen will: xxx.o, Größe 0 Bytes und der Guru kommt in beiden Programmen. Ursprünglich wollte ich das ganze nur neu Übersetzen, aber dann habe ich einen Blick in die Sourcen geworfen. Es war einfach weniger Auf- wand das ganze neu aufzusetzen und Teile zu übernehmen als das alte Programm anzupassen. Ansich ging es ohnehin nur um das Format der .Lib- und .Dir-Dateien.

Anmerkung: Dies ist nicht als 'Flame' an den Originalautor von Libr oder Alib zu verstehen, sondern bezieht sich zum ersten auf die Anpassungen des Sources an vbcc und zum zweiten war der Aufbau einiger Dinge wesentlich größer und bauschiger als nötig. Und soetwas dann anzupassen, sich in eine fremde Programmierung zu denken etc. kostet mehr Zeit als es neu zu schreiben, vor allem wenn man grundsätzliche Funktionen bereitliegen hat und nur zu laden braucht bzw. eine umfangreiche Link-Lib mit Modulen bereit steht. Den HELP- Schirm habe ich aus Alib übernommen und angepasst. Wobei, hätte ich den Source von alib einen Tag früher gefunden dann wäre es wohl nur zu einer Erweiterten alib-Version gekommen. In Alib wurde auf die Option X (Sort der Lib für One-Pass-Linker) verzichtet. Ich schieße mich mit PmmLibr an, weil ich nicht annehme, das sowas noch benutzt wird. Falls doch, kann ich das im Falle eines echten Bedarfs auch jederzeit einbauen.

1.16 Depend-Scan

"PmmScan - Der Depend-Scan (Include-Finder)"

Aufruf: PmmScan <pfad> <programm>

Er wird von Pmm bzw. PmmCmp je nach Einstellung bereits gestartet. Ein manueller Aufruf ist jedoch ebenfalls möglich

Der Pfad gibt das Verzeichnis an, wo sich die Quellen befinden. Das Programm ist der Name unter dem das Programm in der Projekt- datei abgelegt wird. Sinnvoll ist es den Namen zu verwenden, den das Endprogamm erhält. Kann aber auch beliebig anders vergeben sein. Wichtig ist nur, das der Name hier mit dem in der Projekt- datei identisch sein muß.

1.17 Pmm-Funktionsbibliothek

"PmmTool - Eine kleine Funktionssammlung im Aufbau"

PmmTool ist ein Programm, das die Erstellung von Batchfiles vereinfachen soll. Es enthält typische Funktionen, wie sie speziell im Bereich der Programmierung benötigt werden. Im Gegensatz zu vielen Einzelprogrammen, die jeweils eine andere Bedienung benötigen ist hier immer die gleiche Syntax gegeben.

Zur Zeit ist das Tool im Aufbau, es kann erst einen Teil dessen, wofür es gedacht ist.

Aufruf:

PmmTool [-FKT] [PAR1] [PAR2]

-FKT sind:

-c = Compare/Vergleich: Prüft ob die Dateien PAR1 und PAR2 gleich sind. Falls ja, wird das Programm mit 0 beendet, falls nicht, mit 10 (WARNING) und kann so zur Steuerung von Batches benutzt werden. Das fehlen einer Datei ist identisch mit ungleich. -C = Wie vor, aber mit dem Unterschied, das der Buchstabe in klein einen Quiet-Modus ohne Texte, der in groß einen Modus mit Texten bewirkt. Die Returncodes werden in beiden Fällen ausgelöst, nur das -C anzeigt wenn z.B. eine Datei fehlt, ob die Größe verschie- den war oder der Inhalt, was man am Returncode selbr nicht unter- scheiden kann.

-n = IsNewer/Ist neuer: Prüft, ob PAR2 neuer ist als PAR1. Falls ja, wird als Returncode Null zurückgeliefert, falls nicht 10. Das Fehlen einer Datei ist identisch mit "nicht neuer". -N = Wie oben, Textanzeigen aktivieren.

Zukunft:

Es kommen weitere Funktionen in das Programm sowie ein Handlungsparameter der z.B. COPY sein kann und dann im Fall RETURNCODE=10 die Datei kopieren würde. Auch direkte Kombinationen der Funktionen sollen folgen.

Außerdem sollen weitere kleine Funktionen bei Bedarf integriert werden.

1.18 Pmm-Anzeigeprogramm

"PmmView - Eine Oberfläche, wenn's mal groß ist"

PmmView ist der Beginn einer Oberfläche für das System. In diesem Fall ist es das Programm, das die Ausgaben liest, auswertet und dann strukturiert anzeigt. Es ist nicht für einen direkten Aufruf gedacht oder geschaffen sondern wird mittels | (Pipe) verwendet. Also z.B. "vbcc 'paramterliste' | PmmView". Das besorgt PmmCmp, wenn eine der All-Funktionen eingesetzt wird.

Dieses Tool ist zugleich der Grund, warum dies alles BETA heißt. Es kann schon einige Dinge, aber alles noch etwas, na ja, sagen wir 'ungeschickt' oder 'unschön'. Da der nächste Part (der Anzeigeschirm) noch in Arbeit ist, ich aber einen Anhaltspunkt in Sachen Platzbedarf etc. benötigt habe, habe ich eine mögliche Anzeige per Ansi im CLI simuliert. Das klappt, wenn der CLI seine 60-70 Zeichen breit und ca. 20 Zeilen hoch ist, aber es läuft nicht perfekt.

Hier ist die Abhilfe aber schon in Arbeit.

1.19 ToDo-Liste

"ToDo (Die Reihenfolge ist wild und berücksichtigt keine Prioritäten)"

Hinweis: Dies sind einige Stichpunkte, die ich mir nebenbei gemacht habe. Die Liste ist keineswegs vollständig.

- Localize (engl. Version)

- Intui-Oberfläche fertigstellen - Oberfläche muß immer ein PLUS, kein MUSS sein - Kleinere Teiloberflächen für die schlecht zu überblickenden Aufgaben.

- Vernüftige Variablennamen zulassen

- Meldungen der Programme verbessern

- Scripte für AZTEC, DICE, GCC/GNU/gg, E, Pascal usw. - Selber machen oder von Nutzern "einziehen" ;-)

- Editor-Start im Fehlerfall, möglichst mit Fehlerpositionierung

- Voreinstellung für lange Projektcompilings in Abwesenheit: - Stop on Error - Ignore (derzeit, Logging ist noch Schrott)

- Fehlererkennung erweitern (Assembler/Linker) und aus einer Voreinstellung einlesen.

- Source-Verzeichnis mit Datei ".pmm" (als Alternativ-Version zur Projekt-Datei)

- Mehr als ein Projekt, zum einen um ein Projekt evtl. in Gruppen zu unterteilen und zum zweiten, um tatsäch unabhängige Projekte zu gestalten

- Schaltbar: Scannen des Source-Headers. Bestimmte direktiven im Source sollen für das einzelne File Ausnahme-Config's auslösen. Also etwas ähnliches wie die Batch-Header bei SAS/C.

- Work_List -> MakeFile Konverter für Portierungen bei denen Pmm auf dem Zielrechner nicht vorhanden ist oder auch für eine Veröffentlichung bei der man, schon aus Prinzip, von allgemeinen Standards ausgehen muß.

- PmmView: Output auf Oberfläche oder Direkt-CLI, die jetzige Halb- und Halb-Lösung beenden.

- verteilte Source-Codes (innerhalb eines Programmes mehrere Source-Dirs)