

1. Einleitung

Was ist *yab*?

Der yab Interpreter ist ein BASIC Dialekt, welcher eine Weiterentwicklung des auf Windows, Linux und Playstation2 erhältlichen YABASIC ist. Yab wurde für BeOS, Haiku und ZETA optimiert und enthält daher viele zusätzliche Befehle. Sie haben die Möglichkeit mit ein paar Befehlen eine ansprechende Benutzeroberfläche für ihre Programme zu entwickeln. Auch komplexere Aufgaben lassen sich mit YAB erledigen. Im Anhang finden Sie eine Liste mit Programmen die mit YAB erstellt worden sind. Des weiteren ist es möglich das Sie yab selbst erweitern.

1.1 Abkürzung die während des Buches auftreten können.

Abkürzung deutsch	Abkürzung English	Bedeutung
LMT	LMB	linke Maus Taste
MMT	MMB	Mittlere Maus Taste
RMT	RMB	Rechte Maus Taste

2. Installation von YAB

Es gibt jetzt mittlerweile mehr als 2 Möglichkeiten yab zu beziehen und zu installieren. Die erste wäre es über Haikuware zu beziehen. Desweiteren können Sie von der BeSly beziehen als ein sehr einfaches Paket. Eine weitere Möglichkeit wäre, Sie beziehen YAB und den YABCC als ein Paket von der Besly, weil da alle Versionen von YAB enthalten sind und sie Handisch nichts mehr verlinken müssen.

Die Installation von YAB erfolgt mit dem Package von der BeSly sehr einfach.
Sie ziehen sich das Package von der Webseite und klicken danach auf Installieren. Die Installation erfolgt automatisch. Sie müssen nach der Installation sich noch das Paket Ncurses von der oben genannten Webseite ziehen, da in diesem Paket eine dringend benötigte Library liegt. Es gibt schon Mittlerweile auf eine Version ohne Ncurses, welche Sie verwenden, bleibt ihnen überlassen. Sollten Sie aber die Version mit NCurses verwenden, sollten Sie beim installieren des NCurses Paketes nicht automatisch alle überschreiben lassen, da das sich negativ auf andere Programme auswirken kann.

Haben Sie yab ordnungsgemäß installiert, können Sie die yabIDE mit Doppelklick auf die Datei starten.

Wenn Sie yab über das von der Besly zur Verfügung gestellte Paket installiert haben, so haben Sie speziell bei der ZETA Version einen Link in das *Entwicklungsverzeichnis* im ZETA Menü (*Software/Entwicklung*) über die Sie die IDE einfach starten können.

Um Yab direkt von der Entwicklerseite herunter zu laden und zu installieren, lesen bitte bei 2.1 Yab über CVS ziehen und installieren weiter.

2.1 Yab über CVS ziehen und installieren

Um die aktuellste YAB Version direkt aus dem Internet herunter zu laden und immer auf den aktuellsten Stand zu bleiben gibt es das Programm CVS mit dem man den YAB Quellcode herunterladen kann und anschließend kompilieren kann.

CVS was ist das eigentlich ?

Concurrent Versions System (CVS) bezeichnet ein Programm zur Versionsverwaltung von Dateien, hauptsächlich Software-Quellcode. CVS ist ein reines Kommandozeilen-Programm. CVS erfreute sich besonders in der Open-Source-Gemeinde großer Beliebtheit. So wurde CVS bei den meisten großen Open-Source-Projekten verwendet. Die CVS-Software wird unter anderem auch auf den Servern von SourceForge.net verwendet.

Um nun YAB per CVS, das allererste Mal herunter zu laden muss man zunächst über ZETA ->Software -> Systemprogramm -> Terminal, ein Terminal starten.

In das Terminal trägt man folgendes ein:

```
cvs -d:pserver:anonymous@yab-interpreter.cvs.sourceforge.net:/cvsroot/yab-interpreter checkout
yab-interpreter
```

Nach diesem Kommando, das durch betätigen der Enter Taste abgeschickt wird, wird im aktuellen Verzeichnis (meistens: */boot/home*) ein Ordner mit Namen *yab-interpreter* erstellt. In diesem Ordner werden nun einige für YAB benötigte Dateien herunter geladen. Wenn das $ (Dollar) Symbol wieder erscheint und der Cursor blinkt ist CVS fertig.

Nun kann man sich an das Kompilieren von YAB machen.

Achtung!

Das Kompilieren ist nur etwas für erfahrene Anwender, man sollte sich vor dem Kompilieren immer die Anweisungen zum Kompilieren durchlesen.

Im Normalfall liegt der Quellcode im Ordner */src* und die Anweisungen in diesem Ordner unter dem Namen README.Compile

Zunächst einmal sollte man mit dem Kommando *'cd yab-interpreter/src '* in den Quellcode Ordner wechseln.

Mit folgendem Befehl kompiliert man YAB dann.

```
make -f Makefile.ZETA
```

Die ausführbare YAB Datei befindet sich dann im Ordner src. Die kompilierte YAB Datei muss dann in das Verzeichnis */boot/home/config/bin* verlinkt werden, damit ist yab dann installiert.

2.2 Yab aktuell halten

Um Yab über CVS aktuell zu halten, führen Sie bitte im Terminal folgenden Befehl aus.

```
cvs -d:pserver:anonymous@yab-interpreter.cvs.sourceforge.net:/cvsroot/yab-interpreter
update -d yab-interpreter
```

2.3 Wo finde ich danach die YAB-IDE?

Wenn Sie nun yab über *CVS* bezogen haben, finden Sie die IDE in folgenden Verzeichnis:

/boot/home/yab-interpreter/yab-IDE

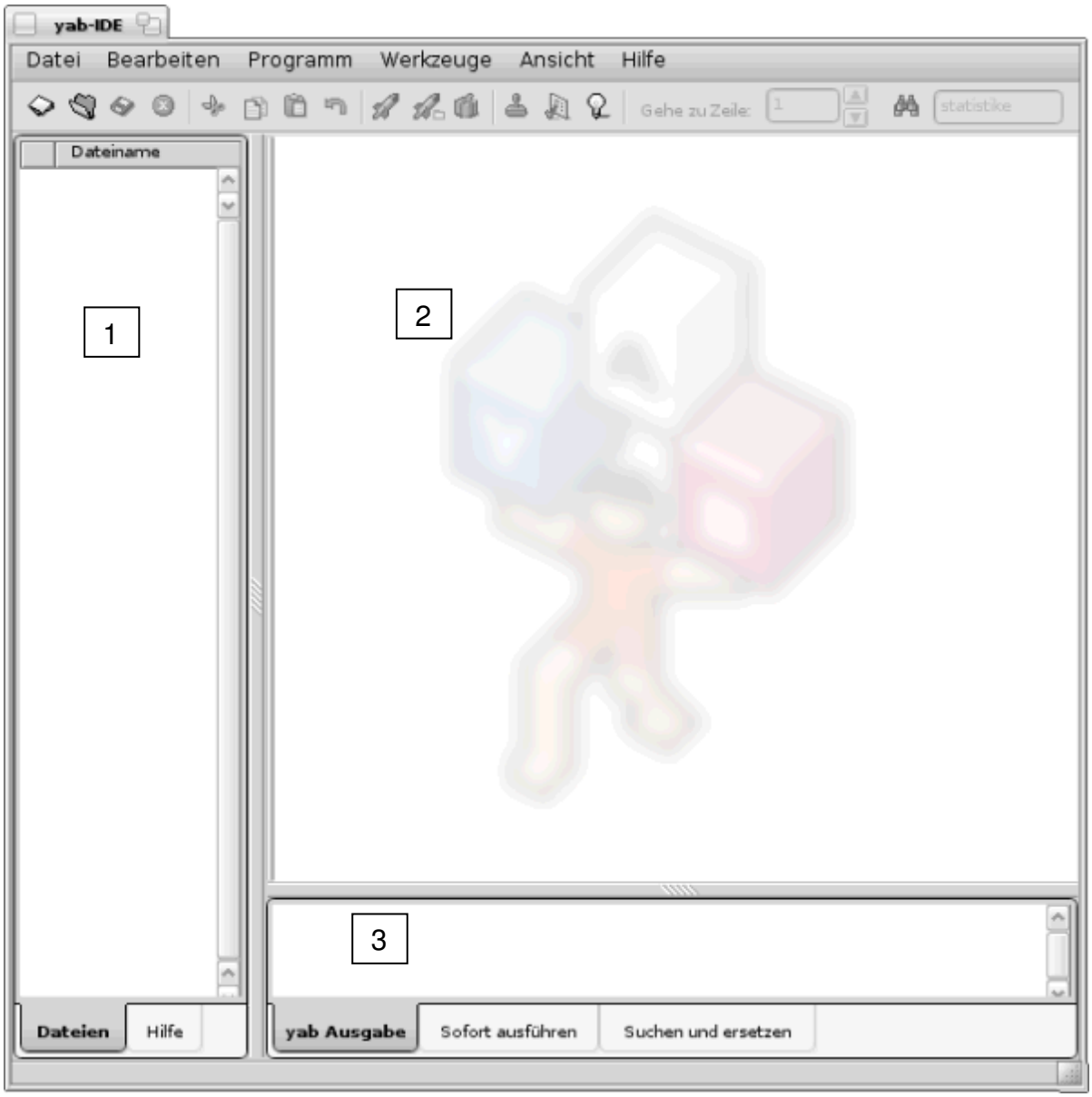
Da die *yabIDE* auf Dateien und Verzeichnisse in diesem Verzeichnis verweist, müssen Sie den yab Ordner umbenennen: *"/boot/home/yab-interpreter"* in *"/boot/home/yab-1.0"*.

Im ersten Moment erscheint es unsinnig das man den Ordner umbenennen muss, hat aber einen guten Grund, denn so trennen Sie die herunter geladene Version von einer eventuell später neu bezogenen Version. Dadurch haben Sie die Sicherheit, dass keine Daten verloren gehen.

Da die yabIDE in yab geschrieben ist und als reine Sourcedatei zur Verfügung steht, könnten Sie jetzt natürlich auch bei gehen und den gesamten Sourcecode durchwandern und sämtliche Pfadangaben ändern.

3. YAB IDE Grundlagen

Die *yabIDE* ist eine Entwicklungsumgebung für *yab*. Der Editor besticht durch seine Funktionen. Es ist ihnen möglich ihre Programme darin zu Schreiben und auch sofort zu testen. Zudem kommt auch noch das automatische hervorheben (Syntax Highlighting) von YAB Befehlen. Um ihnen schnell eine Übersicht zu verschaffen, beschreiben ich hier kurz die Bereiche in der yab-IDE.



Startbildschirm von der yab-IDE

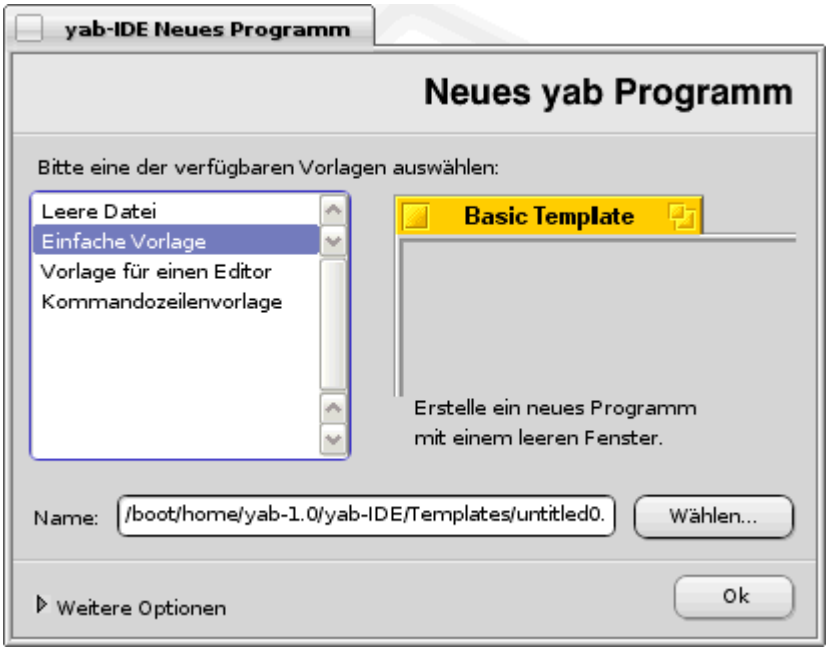
1 In diesem Teil der yab-IDE wird ihnen die Hilfe oder die aktuell von ihnen bearbeiteten Dateien angezeigt.

2 In diesem teil der yab-IDE sehen Sie ihren Programmcode

3 In Diesem Teil der yab-IDE sehen Sie print Ausgaben oder Fehlermeldungen. Über den zweiten Tab können Sie befehle sofort ausführen und über den dritten Tab suchen und ersetzen.

3.1 Vorlagen laden

Wer das erste mal mit *yab* zu tun hat, wird sicherlich gerne auf in der IDE vorhandene Vorlagen zurückgreifen.



Um das Auswahlfenster zu öffnen, gehen Sie im yab-IDE Menü auf *Datei* und dort auf *Neu...*. Hier haben Sie die Möglichkeit aus mehreren Vorlagen eine für Sie entsprechende auszuwählen.

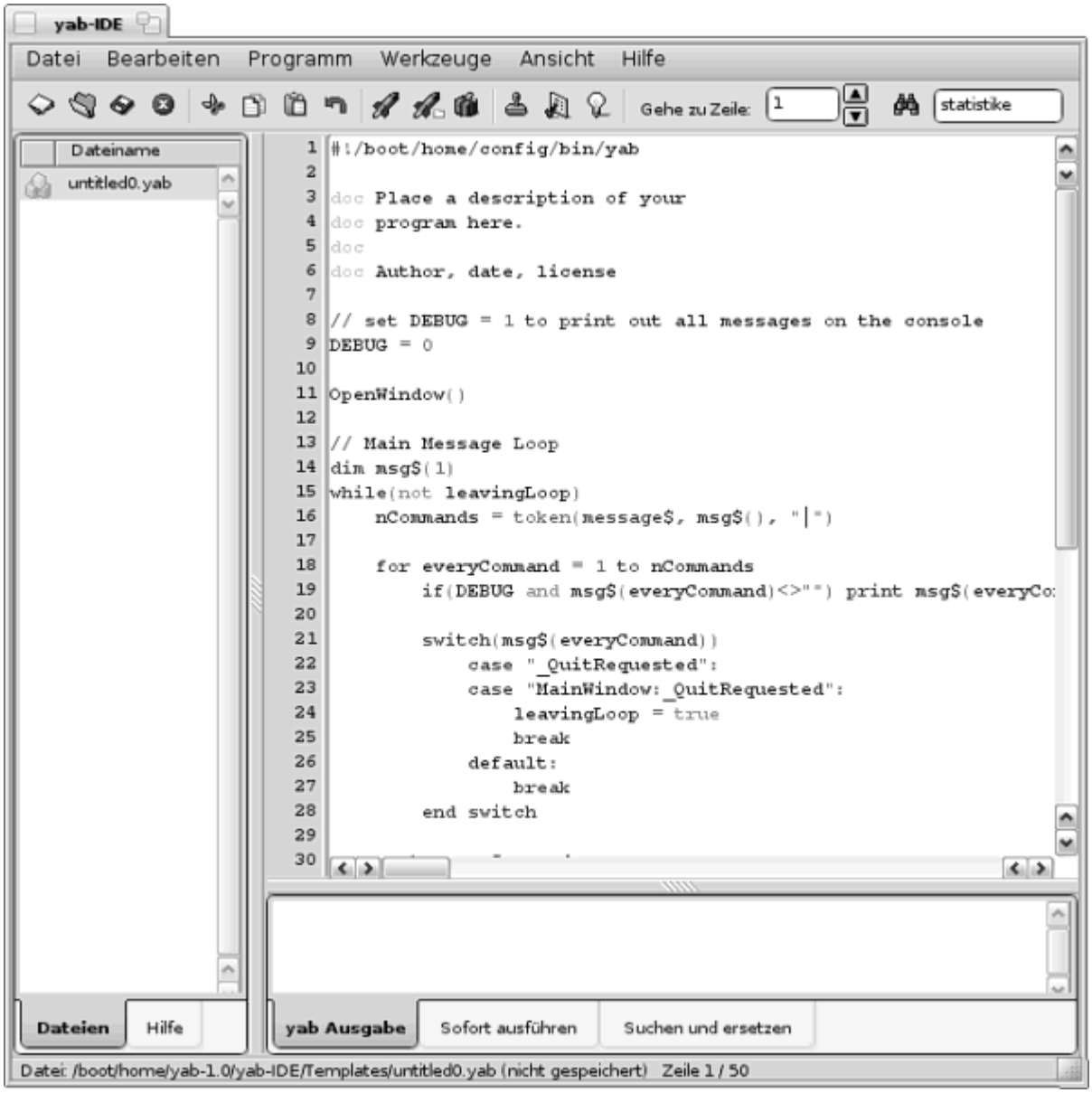
Entscheiden Sie sich für eine *Leere Datei* erhalten Sie eine, wie sollte es anders ein, *leere* Seite. Dies bringt einem unerfahrenen und gerade erst beginnenden yab Programmierer natürlich bedeutend wenig, daher ist hier eine andere Auswahl mit Sicherheit zu empfehlen.

Da wäre einmal die *Einfache Vorlage*, welche einen Programmcode beinhaltet, welches ein einfaches Fenster erstellt. Funktionen wie zum Beispiel das schließen des Fensters, sind bereits mit einbracht. Eine weitere Vorlage bietet alle Funktionen für das erstellen eines Editors (*Vorlage für eine Editor*).

Es sind eventuell nicht alle Vorlagen in diesem Auswahlmenü aufgeführt, daher können Sie auch einfach mal im *Datei-* Menü mit *Öffnen...* im */boot/home/yab-1.0/yab-IDE/Templates* Verzeichnis nach weiteren Vorlagen gucken.

3.2 Zeilennummern anzeigen

Eine Funktion die bei vielen Editoren fehlt ist glücklicherweise in der yab-IDE enthalten. Die Rede ist von der Anzeige der Zeilennummern, welche eine schnelle Übersicht über den derzeitigen Standort im Programmcode ermöglicht.



Wenn Sie die IDE starten, werden, wenn Sie diese nicht vorher bereits angezeigt haben, die Zeilennummern nicht dargestellt. Diese verbergen sich hinter dem linken Rand des Editierbereiches. Klicken Sie auf den dort stehenden dünnen Balken und ziehen, während sie die Maus gedrückt halten, diese nach rechts. In der BeSly Version können Sie unter Programm -> Einstellungen auch so einstellen, dass die Zeilennummern immer angezeigt werden.

3.3 Die Werkzeugleiste

Die Werkzeugleiste (Toolbar) beinhaltet die wichtigsten Funktionen, die man während der Programmierung immer wieder braucht. Die folgende Tabelle erläutert deren Funktion:

- Öffnet das Vorlagen Auswahlmenü
- System nach einer vorhandenen Datei durchsuchen und öffnen.
- Programmcode abspeichern.
- Datei schließen
- Markierten Bereich ausschneiden
- Markierten Bereich in die Zwischenablage kopieren.
- Inhalt der Zwischenablage einfügen.
- Vorherigen Arbeitsschritt zurücksetzen.
- Programmcode intern der IDE ausführen.
- Programmcode im Terminal ausführen.
- BuildFactory öffnen um Sourcecode als C++ Code zu kompilieren.
- Muster für yab Befehlt *Palette* erstellen.
- yabIDE Einstellungen vornehmen.
- yabIDE Hilfe aufrufen.

Gehe zu Zeile: Geben Sie die gewünschte Zeilenummer über Tastatur oder mit den Pfeiltasten ein und drücken die *Eingabetaste* (*Return, Enter*).

Geben Sie einen Suchbegriff in das Eingabefeld ein und drücken die *Eingabetaste* (*Return, Enter*) oder das *Fehrnglas* Symbol.

3.4 Tastaturkürzel

Tastaturkürzel oder besser verständlich als Tastenkombinationen sind für einen erfahrenen Programmierer ein wichtiger Bestandteil eines Editors. Nicht nur das, dass ewige klicken auf Schaltflächen, umständlich und zeitraubend ist, wer eh mit der Tastatur beim Eingeben von Programmcodes beschäftigt ist, kann eine gewünschte Funktion mit der Tastatur bedeutend einfacher ausführen.

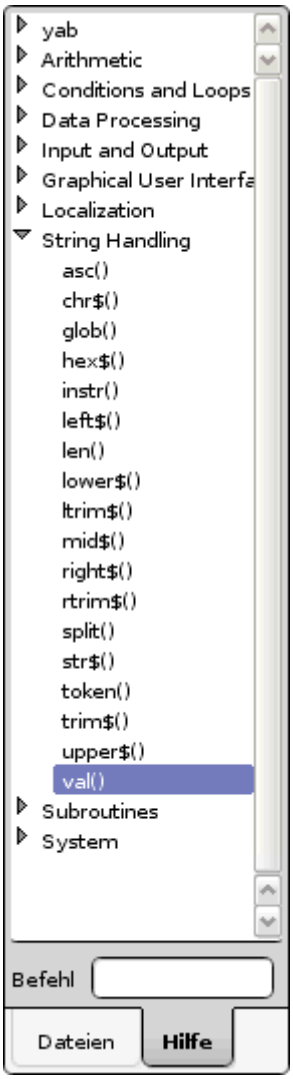
Die meisten Tastenkombinationen sind ähnlich wie die des Systems. Einige davon sind jedoch speziell für Funktionen der IDE zuständig, daher werden wir hier einmal die wichtigsten aufführen.

Alt + N	Öffnet Vorlagen- Menü
Alt + O	Öffnet Filepanel zum auswählen einer Datei.
Alt + C	Kopiert einen markierten Bereich in die Zwischenablage.
Alt + V	Fügt in der Zwischenablage befindenden Code ein.
Alt + X	Markierten Bereich ausschneiden.
Alt + S	Speichern des geöffneten Programmcodes
Alt + Z	Arbeitsschritt rückgängig machen (*Undo*).
Alt + W	Geöffnete Datei schließen
Alt + R	Programmcode intern der IDE ausführen.
Alt + T	Programmcode über Terminal ausführen.
Alt + F	Im Suchenfeld eingetragenen Begriff suchen.
Alt + G	Von ersten gefundenen Begriff zum nächsten wechseln (Weitersuchen).
Alt + E	Öffnet *Suchen und Ersetzen* Eingabebereich.
Alt + A	Alles auswählen.

Wer mehr Tastenkombinationen erfahren möchten, kann diese auch im Menü der IDE hinter den einzelnen Einträgen sehen.

3.5 Hilfe

Die yabIDE beinhaltet eine umfangreiche Hilfe. Diese beinhaltet alle möglichen Befehle für yab, mit Beschreibung und Beispielcodes. Einträgen sehen.



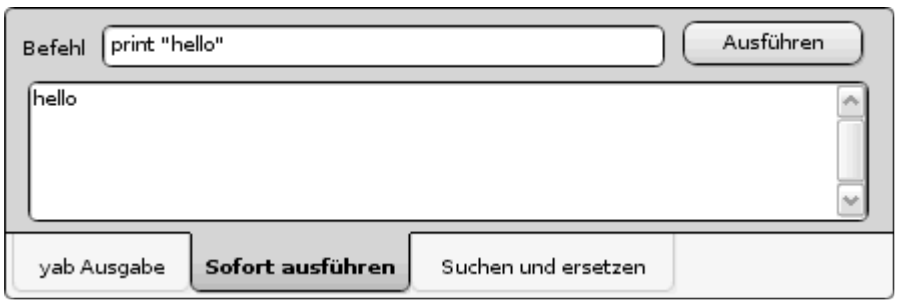
Die Hilfe wird aufgerufen, in dem Sie den Reiter (TAB) *Hilfe* im unteren linken Teil der IDE anklicken. In dem Bereich, in dem vorher Ihre geöffneten Programmdateien aufgelistet waren, wird nun die Hilfe angezeigt. Wenn Sie nun einen der dort aufgelisteten Einträge anklicken wird in dem Bereich in dem vorher Ihr Programmcode gestanden hat, der Hilfetext ausgegeben.

Der Hilfebereich beinhaltet auch eine Suchfunktion. Um einen bestimmten Befehl zu suchen, geben Sie im Eingabefeld *Befehl*, einen solchen ein und drücken die Eingabetaste. Es wird direkt in die jeweilige Hilfedatei gesprungen, das Hilfemenü wird dazu leider nicht angepasst.

Um zurück zu Ihren Programmcode zu kommen, wechseln Sie den Reiter (TAB) wieder auf *Dateien.*

3.6 Befehle sofort ausführen

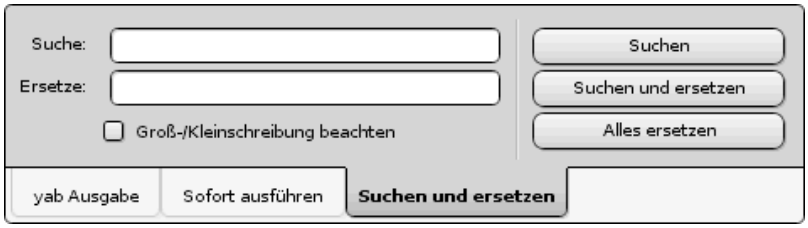
Im unteren Teil der IDE befinden sich drei Reiter (Tabs). Der mittlere davon dient dazu, yab Befehle direkt auszuführen.



Um einen Befehlt direkt auszuführen, geben Sie diesen einfach in das Eingabefeld *Befehl* ein und drücken die Eingabetaste. Die Ausgabe des Befehls wird im darunter liegenden Ausgabefeld getätigt.

3.7 Suchen und ersetzen

Wenn Sie einen Befehl oder einen Begriff (Textbereich) suchen und durch einen anderen Ersetzen möchten, wechseln Sie im unteren Bereich der IDE auf den Reiter (TAB) *Suchen und Ersetzen*.



Geben Sie im Eingabefeld *Suchen* den Suchbegriff, und im Eingabefeld *Ersetzen* den Begriff ein, womit Sie das Vorhandene ersetzen möchten.

Nun haben Sie mehrere Möglichkeiten, zum einen können Sie nur den Suchbegriff suchen, indem Sie die Schaltfläche *Suchen* betätigen. Das gefundene Wort wird im Programmcode markiert. Drücken Sie erneut auf *Suchen*, wird automatisch zum nächsten Begriff weitergesprungen.

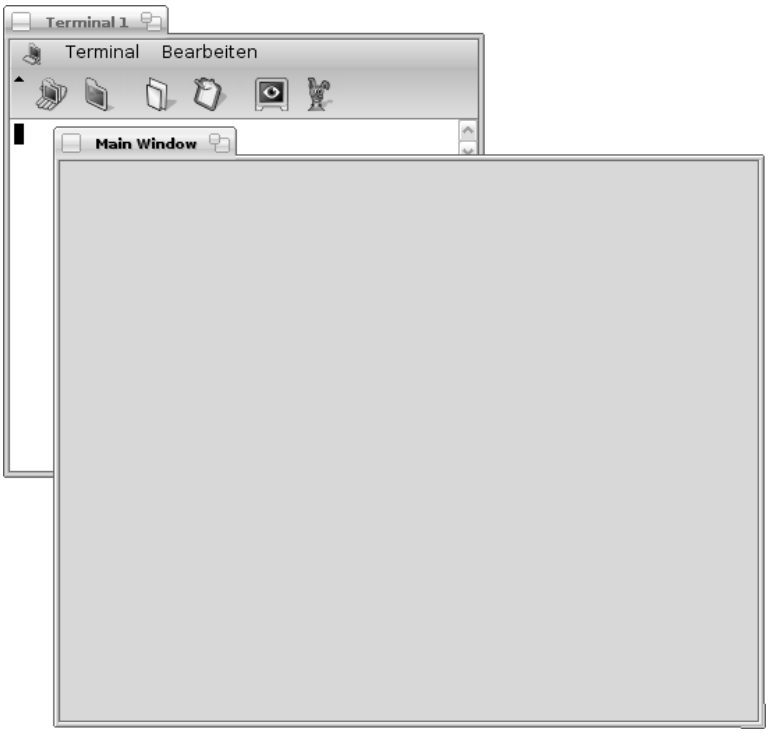
Dann haben Sie die Möglichkeit einen Begriff zu Suchen und zu Ersetzen, indem Sie auf die Schaltfläche *Suchen und Ersetzen* klicken. Auch jetzt wird der gefundene Begriff im Programmcode markiert. Durch erneutes drücken auf *Suchen und Ersetzen* wird wie bei der Suche auch zum nächsten Begriff gesprungen, nur das dabei dieser gleich durch den neuen Begriff ersetzt wird.

Die dritte Möglichkeit ist das Ersetzen aller Begriffe im Programmcode. Drücken Sie auf die Schaltfläche *Alles ersetzen*, werden sämtliche gefundene Begriffe sofort ersetzt.

Wenn Sie die Suche genauer einstellen möchten, können Sie diese so einstellen, dass dabei auf Groß- und Kleinschreibung geachtet wird. Setzen Sie dazu einen Haken bei *Groß-/Kleinschreibung beachten*.

3.8 Programmcode ausführen

Der Vorteil bei yab ist sicherlich die Möglichkeit, den bereits geschriebenen Programmcode, jederzeit ausführen zu können, um zu sehen wie etwas aussieht oder ob eine programmierte Aufgabenstellung funktioniert.



Die yab-IDE bietet dafür zwei Möglichkeiten. Zum einen kann man den Programmcode über den Terminal ausführen und zum anderen direkt über die IDE. Bei beiden Varianten bekommt man, wenn man einen diesbezüglichen Bereich im Programmcode vorsieht, Meldungen über die durchgeführten Bedienschritte des Programms ausgegeben (*Siehe zweites **Einsteiger Tutorial***).

Um den Programmcode im *Terminal* auszuführen, klicken Sie in der Werkzeugleiste (Toolbar) auf das

Raketensymbol mit dem kleinen Fenster  oder mit der Tastenkombination *Alt +T*.

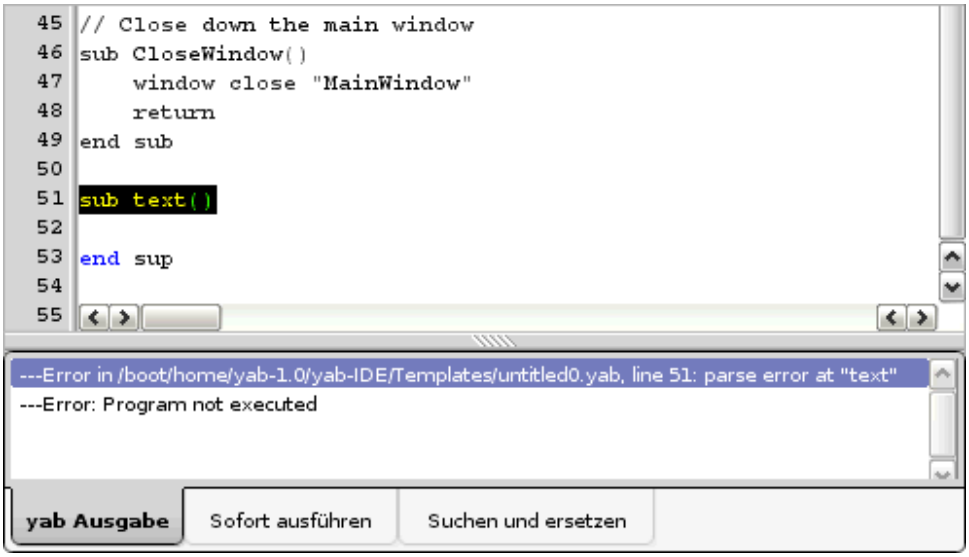
Das Programm wird in einem Terminalfenster gestartet. Fehlermeldungen oder Probleme die beim Ausführen auftreten können, werden dabei im Terminal ausgegeben. An Hand dieser Fehlermeldungen kann man dann im Programmcode diese lokalisieren und beheben.

Um den Programmcode direkt in der yab-IDE auszuführen, klicken Sie in der Werkzeugleiste (Toolbar) auf das Raketensymbol  oder mit der Tastenkombination *Alt +R*.

Die Ausführung vom Programmcode in der IDE hat da eine entscheidenden Vorteil, denn hier wird eine Fehlermeldung oder Problem als Verknüpfung in die jeweilige Zeile des Programmcodes ausgegeben. Dies macht ein lokalisieren eines Fehlers oder Problems bedeutend einfachen.

3.9 Fehlersuche

Treten Fehler auf beim Ausführen des Programmcodes, werden diese, je nachdem wo man diesen ausführt, im Terminal oder im Ausgabereich der IDE ausgegeben.



Die Ausgabe der Fehlermeldung ist in beiden Fällen identisch, der schon vorher erwähnte Vorteil beim Ausführen des Programmcodes in der IDE wird Ihnen jedoch am meisten behilflich sein, den Fehler zu lokalisieren. Dazu mache Sie einfach einen Doppelklick auf die Fehlerausgabe, wodurch Sie automatisch in die Zeile springen, in der dieser sein soll.

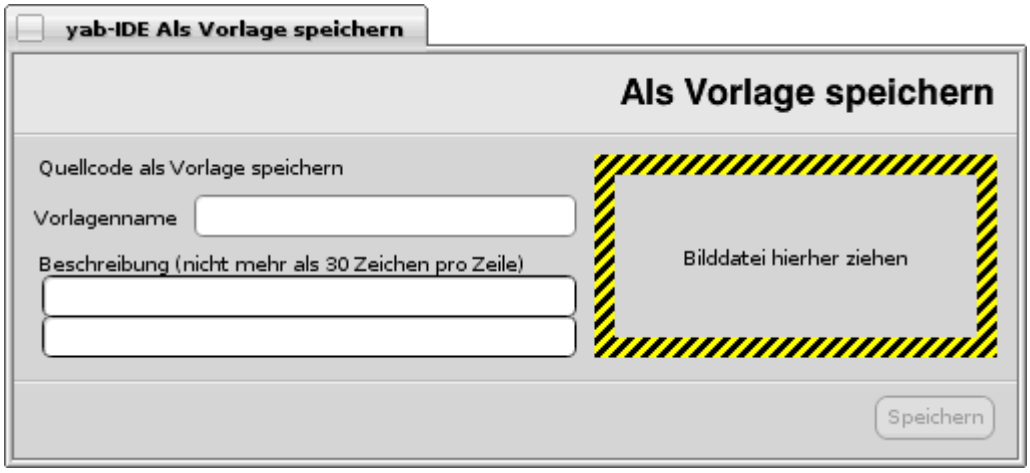
In der Fehlerausgabe steht auch die nötige Information, um zu erfahren, wo der Fehler liegt. Der Fehler in der Abbildung liegt darin, dass hier der Abschlussbefehl für die Subroutine falsch geschrieben wurde. Des Weiteren werden Reservierte Begriffe in der IDE Blau dargestellt, z.B. Sie wurden das Wort sub als Variable verwenden, obwohl dies ein Befehl für die Erstellung einer Unterroutine ist.

Gern gemachte Fehler sind zum Beispiel das öffnen einer Schleife, welche dann am Ende nicht zurückgesetzt wird, oder das Starten einer *if* Abfrage, die dann nicht mit *endif* beendet wird. Achten Sie deshalb immer darauf das Sie die Dinge die sie starten auch am Ende wieder beenden oder zurücksetzen.

Die Fehlerausgabe gibt nicht immer den genauen Fehlerort aus, sind zum Beispiel Probleme im Ablauf einer Subroutine aufgetreten, kann es sein das der Fehlerort das Ende der Subroutine *end sub* ist. Können Sie also eine Fehlerquelle nicht direkt ausmachen, gehen Sie alle zuvor ablaufenden Abläufe durch um diesen zu lokalisieren.

3.10 Programmcode als Vorlage speichern

Wenn man immer wieder die selben oder ähnliche Dinge programmiert, ist es sinnvoll, diese als Vorlage abzulegen. Das macht es einfacher, diese wenn man sie benötigt, schnell aufzurufen oder als Grundgerüst zu verwenden. Um ein Programmcode als Vorlage zu speichern, geht man im *Datei*-Menü auf *Als Vorlage speichern...*



Im *Vorlage speichern* Fenster gibt man der Vorlage erst einmal einen Namen bei *Vorlagenname*. Hier sollte ein Name gewählt werden, der die Funktion der Vorlage eindeutig beschreibt, damit diese einfacher zu finden ist.

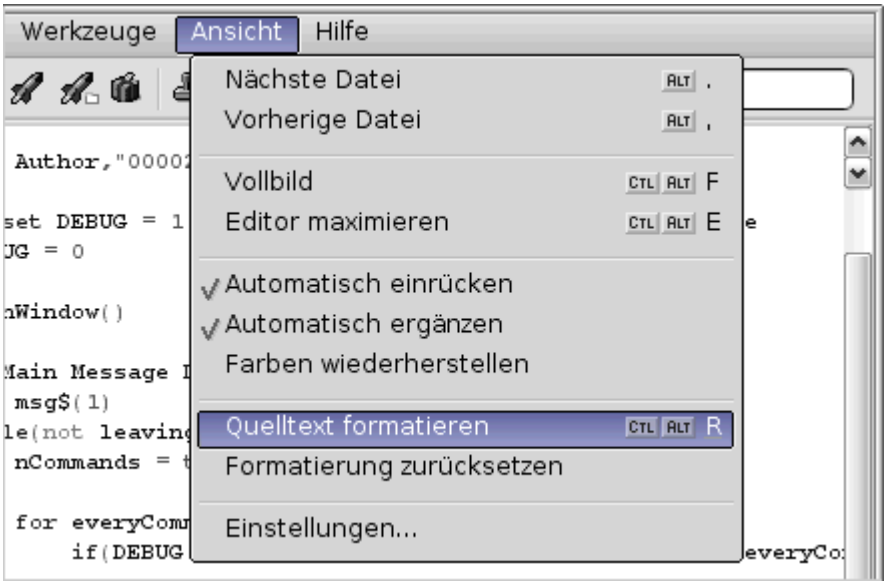
In den beiden darunter liegenden Eingabefeldern geben Sie eine Beschreibung für die Vorlage ein. Sind es mehrere Funktionen die in dieser Vorlage erfüllt werden, sollten diese hier aufgeführt werden.

In dem Gelb- Schwarz gestichelten Kästchen, kann man eine Grafik draufschieben, welche dann als Vorschau für die Vorlage mit angezeigt wird. Die Grafik sollte 200 Pixel in der Breite und 100 Pixel in der Höhe haben.

Sind alle Angaben gemacht, speichert man die Vorlage mit drücken der *Speichern* Schaltfläche. Danach kann die Vorlage über den Menüpunkt *Neu...* aus der Liste mit Vorlagen ausgewählt werden.

3.11 Programmcode Formatieren

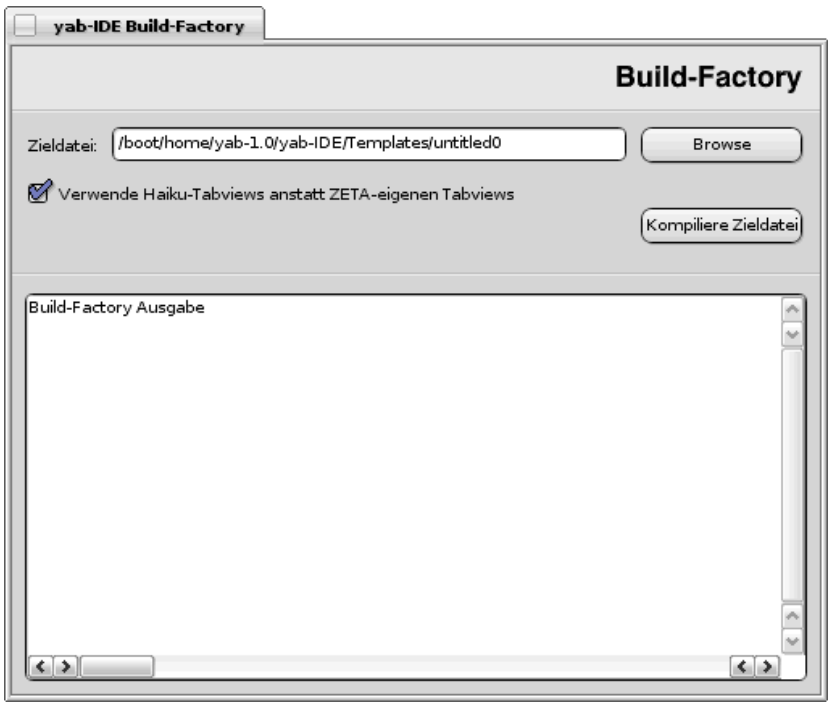
Für diejenigen, die die Angewohnheit haben, Ihren Programmcode chaotisch zu gestalten, hat der Entwickler der yab-IDE eine Funktion zum Formatieren des Pogrammcodes eingebaut.



Die Funktion wird aufgerufen, wenn man im *Ansicht*- Menü auf *Quelltext formatieren* geht. Daraufhin wird der Programmcode so formatiert, dass bestimmte Programmabschnitte eingerückt werden, um eine bessere Übersicht zu schaffen.

3.12 Build Factory

Mit der Build Factory kann man seinen fertigen Programmcode als C++ Programm kompilieren. In den Einsteigertutorials haben Sie Ihren Programmcode immer gebunden. Das kompilieren ist jedoch eine viel schönere Lösung, da dadurch das Programm bedeutend kleiner wird, da yab nicht mit in die Datei eingebunden wird. Das ist auch der Grund, weshalb man über die IDE nur diese Art der Fertigstellung eines Programms hat. Natürlich können Sie immer noch Ihren Programmcode binden, dies aber nur über die Eingabe im Terminal.



Die Build Factory rufen Sie über das *Programm*- Menü - Auswahl *Build Factory* auf.

Um Ihren Programmcode zu kompilieren, wählen Sie bei *Zieldatei:* die Datei aus, welche Sie gerade in der IDE geöffnet haben, also die Datei, welche Sie als C++ Programm kompilieren möchten. Sie müssen die Datei überschreiben. Am besten machen Sie vorher

eine Sicherheitskopie um einen eventuellen Verlust vorzubeugen (derzeit noch nicht vorgekommen).

Wenn der Haken bei *Verwende Haiku- Tabviews anstatt ZETA- eingenen Tabviews* gesetzt ist, werden die Haiku TABs verwendet. Dies ist natürlich nur relevant, wenn das Programm Tabviews beinhaltet.

Nun können Sie mit betätigen der *Kompiliere Zieldatei* Schaltfläche den Programm-code kompilieren. Im unteren Ausgabefeld werden dabei die Schritte des Kompilier-vorganges ausgegeben. Wenn Fehler auftreten, werden diese hier ebenfalls ausgegeben.

Grundsätzliche Regelungen

Um den yab Sourceode als C++ Binary kompilieren zu können, müssen Sie folgendes beachten:

- Namen für Subroutinen dürfen nur einmal verwendet werden, egal wie viele Dateien Sie verwenden. Sollte ein Name doppelt vorkommen, wird nur eine Subroutine berücksichtigt.

- Verwenden Sie Libraries, so achten Sie bitte darauf das in diesen nur *subs* und *export subs* verwendet werden dürfen. Sollten andere Funktionen zwischen den subs und export subs verwenden werden, werden diese beim kompilieren ignoriert.

- Alle Libraries die mit *import* in den Programmcode eingebunden werden, werden berücksichtigt. Dabei werden auch Libraries, die in Libraries eingebunden werden beachtet.

- Die Libraries müssen im selben Verzeichnis liegen wie die Quelldatei oder in */boot/home/config/lib/yab/*.

- Zum kompilieren, müssen alle Quelldateien im *BuildFactory* Verzeichnis liegen.

Die Variante für die Terminal nutzer.

Um jetzt Ihren Sourcecode zu kompilieren, kopieren Sie alle zum Sourcecode gehörenden Dateien in das *BuildFactory* Verzeichnis (Hauptdatei, Libraries, usw.).

Öffnen Sie den *Terminal* und wechseln in das *BuildFactory* Verzeichnis. Nun geben Sie den Befehl zum kompilieren der Quelldatei ein:

```
cd /boot/home/yab-interpreter/BuildFactory
yab BuildFactory.yab [-ZETA-Tab] Ausgabedatei Quelldatei
```

Wird [-ZETA-Tab] mit angegeben, werden die ZETA Tabs verwenden. Lassen Sie diese Angabe weg, werden die Haiku Tabs verwendet (Standard).

Das Binary wird wenn Sie kein Zielverzeichnis mit angegeben haben, im *BuildFactory* Verzeichnis abgelegt. Zusätzlich werden noch weitere Dateien mit abgelegt, einmal der zusammengefasste *neue* Sourcecode und Fehlerausgabedateien.

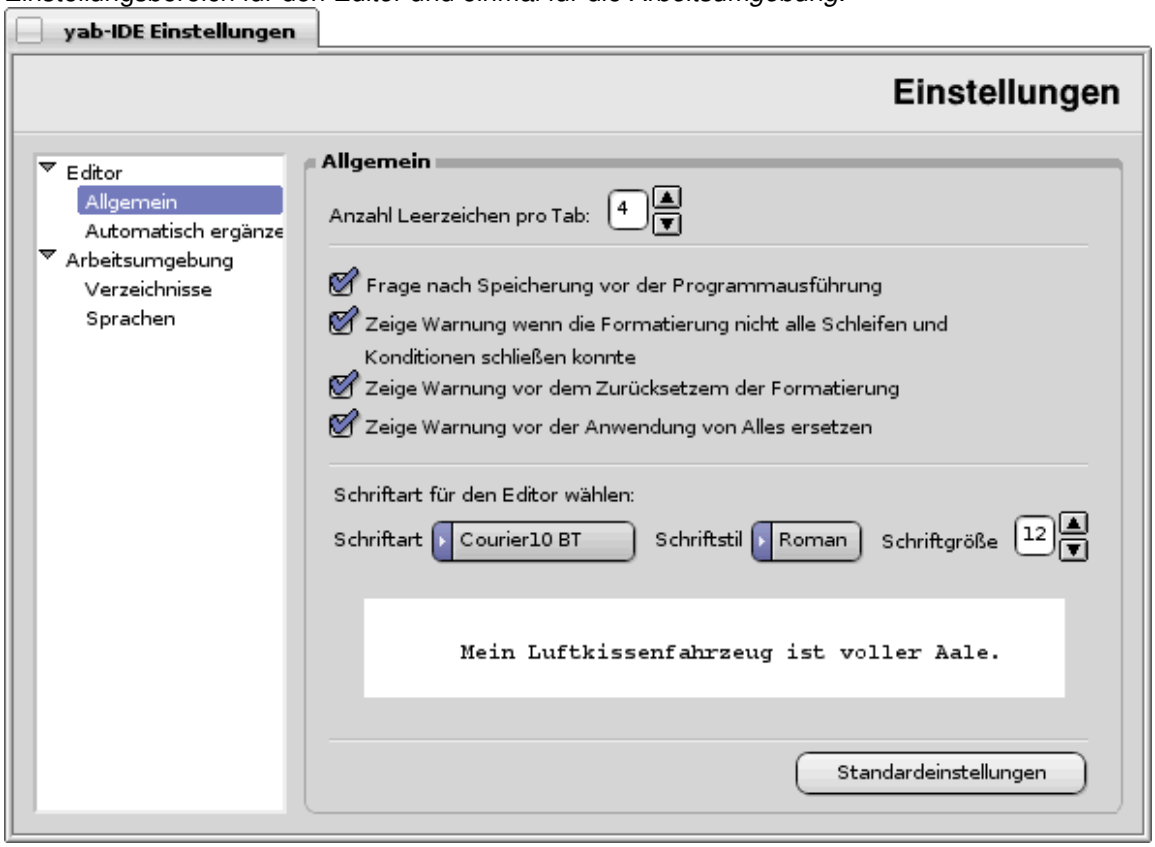
Mögliche Probleme

- Bei älteren ZETA Versionen werden die ZETA Tabs nicht funktionieren, deshalb wird es dann zu einer Fehlermeldung kommen. Verwenden Sie hier einfach die Haiku Tabs.

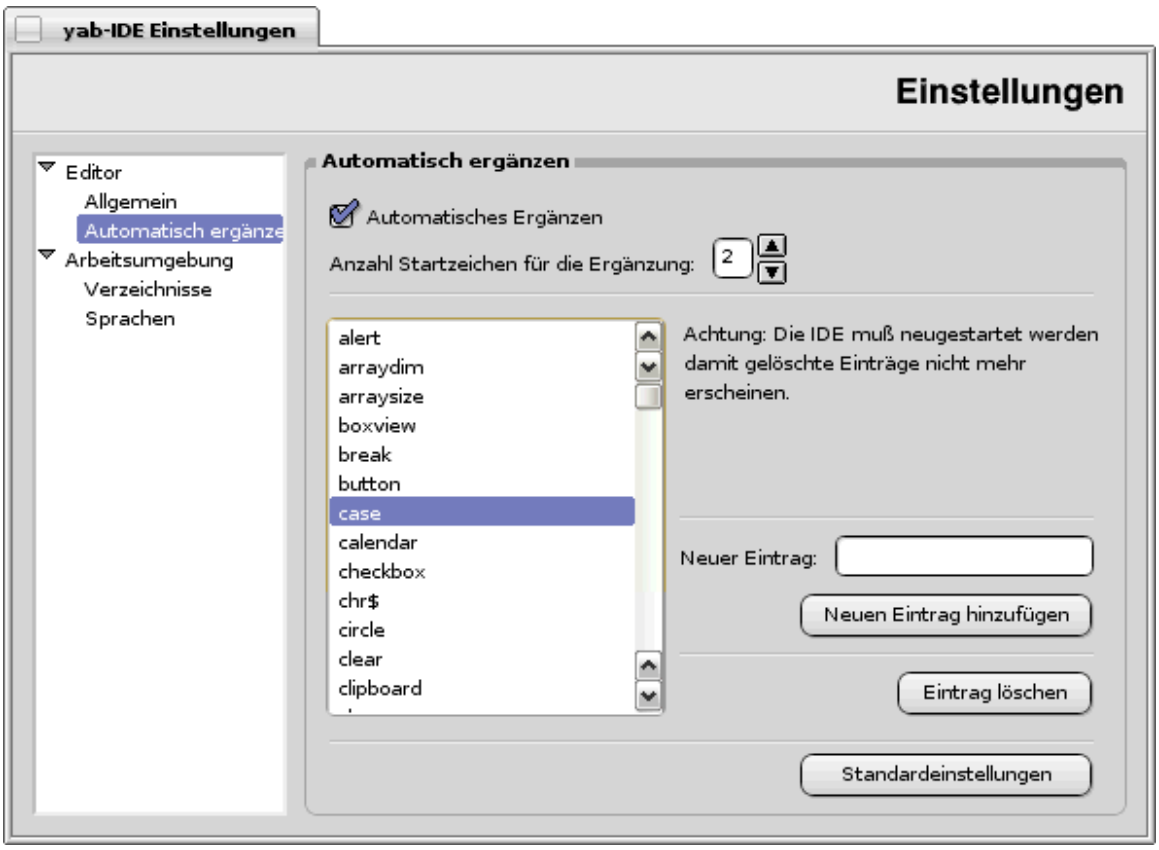
- Sie benötigen unbedingt *ncurses* um Ihren Sourcecode über die BuildFactory kompilieren zu lassen. Sollte *ncurses* nicht installiert sein, können Sie diese hier beziehen.

3.13 Einstellungen

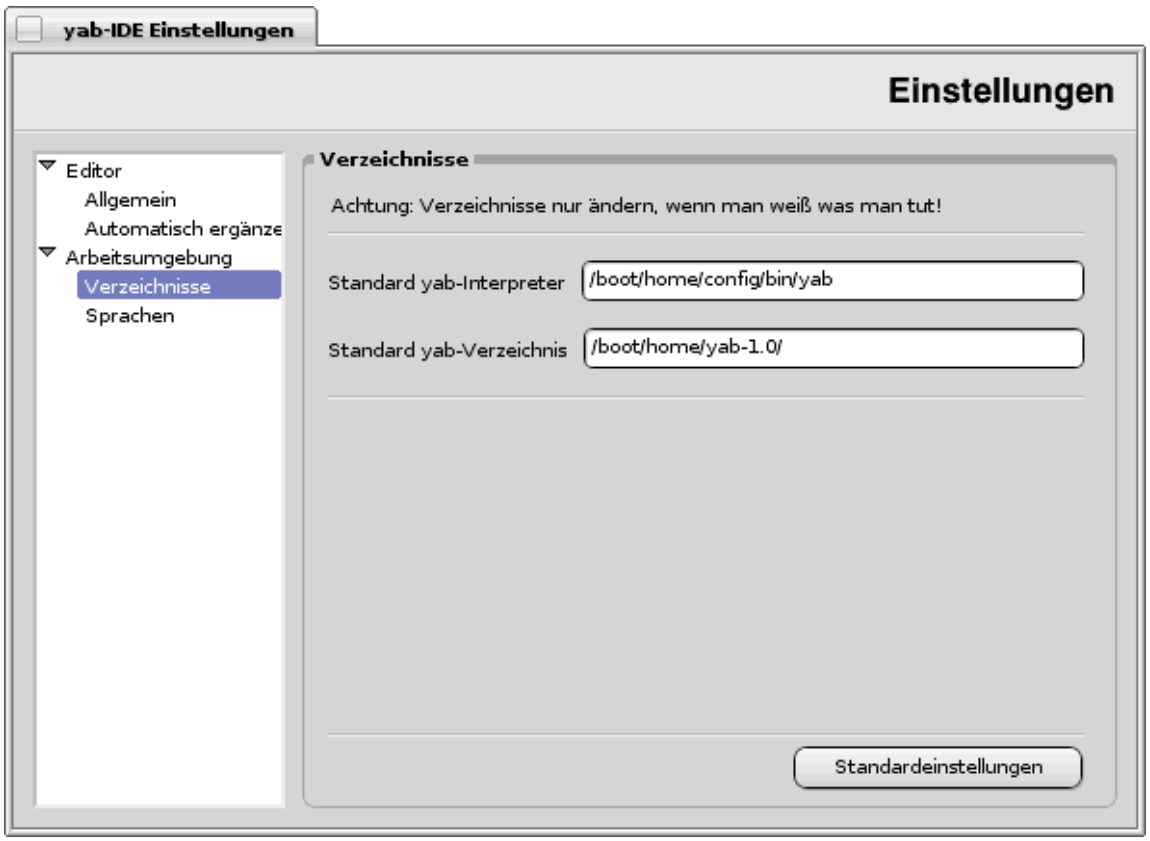
Natürlich gibt es auch einen Einstellungsbereich für die IDE. Diese wird aufgerufen über das *Ansicht*-Menü und der Auswahl *Einstellungen*. Die Einstellungen können nur dann aufgerufen werden, wenn eine Datei geöffnet ist. Die Einstellungen sind in zwei Bereiche aufgeteilt, einmal in den Einstellungsbereich für den *Editor* und einmal für die *Arbeitsumgebung*.



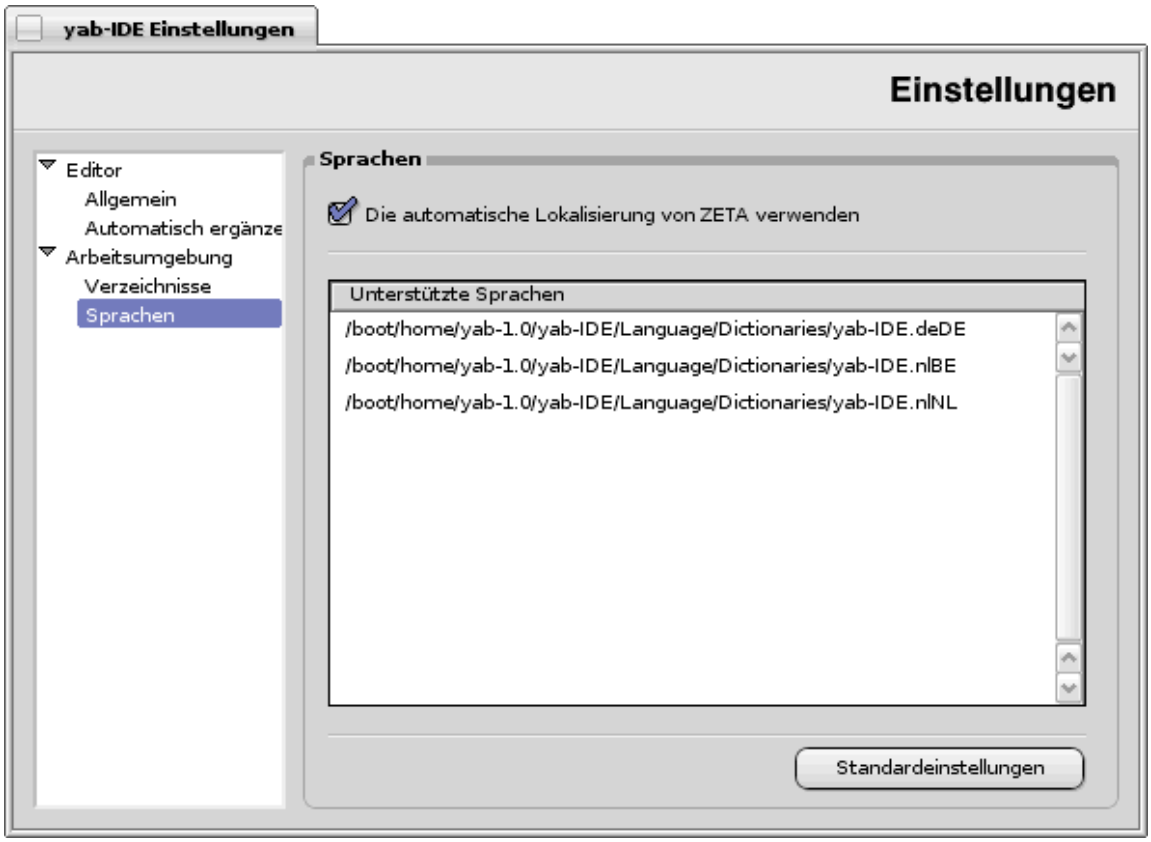
In den Allgemeinen Einstellungen für den *Editor* sind Funktionen wie zum Beispiel dem Nachfragen zum Speichern der Datei, bevor diese zum Ausführen freigegeben wird und der Schriftart für den Editor. Hier werden praktisch alle Funktionen die variabel gehalten werden können aufgeführt und zum Ändern freigegeben.



Ein weitaus interessanterer Einstellungsbereich für den Editor ist der für die *automatische Ergänzung*. Hier können Sie eigene Befehle oder Begriffe eintragen, die dann automatisch ergänzt werden können. Außerdem kann man hier auch angeben, ab welchen Zeichen die automatische Ergänzung tätig werden soll. Steht diese Angabe zum Beispiel auf 2, wird ab den zweiten Zeichen eine automatische Ergänzung durchgeführt.



Bei den Einstellungen für die *Arbeitsumgebung* ist es möglich die Programmverzeichnisse zu verändern. Gemeint sind dabei die Verzeichnisse für das *yab* Binary und für das *yab-1.0* Verzeichnis in welchen sich ja auch die IDE befindet.



Im *Sprachen* Einstellungsbereich kann man das Nutzen des ZETA Lokalkits aktivieren und deaktivieren. Hier werden in einer Tabelle alle enthaltenen Sprachdateien für das Lokalkit aufgelistet.

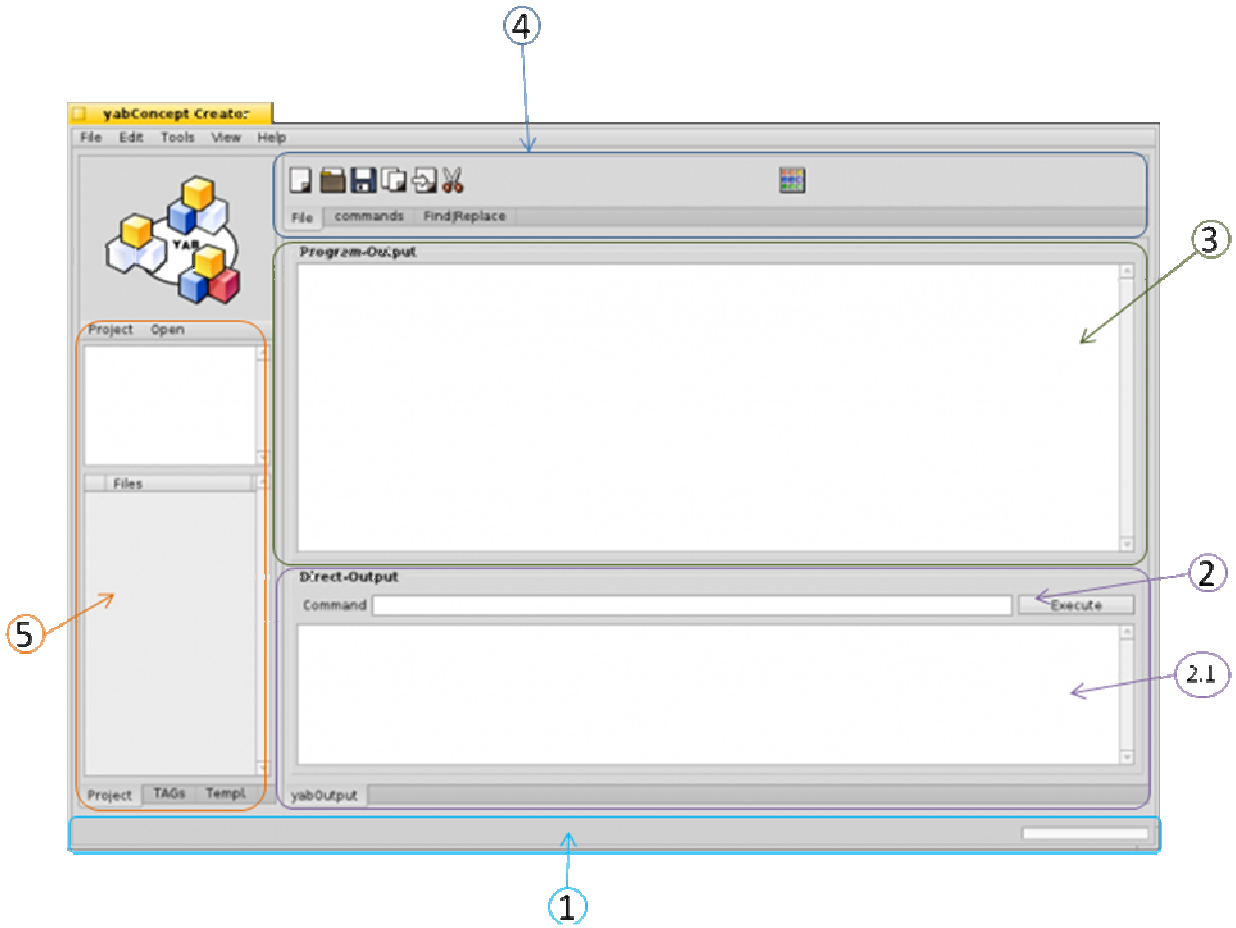
4. Der yabConcept Creator

Der Yab Concept Creator entstand aus eigenem Antrieb. Die beigefügte IDE von YaB ist für den Anfang ausreichend, aber mit der Zeit möchte man einfach mehr Funktionen haben und das Arbeiten mit anderen IDE's für andere Programmiersprachen zeigte uns einfach, das man da eine Menge erweitern kann und sollte.

In diesem Kapitel wird ihnen der komplette Yab Concept Creator erklärt. Dieser besitzt viele Funktionen die auf den ersten Blick nicht zu erahnen sind.

4.1 Installation

Wenn Sie den yabConceptCreator von der BeSly heruntergeladen und die Installation durchgeführt haben, steht ihnen nicht nur der YabCC zur Verfügung, sondern auch die Yab Versionen von 1.0 – bis derzeit 1.7. Nicht alle Versionen haben den gleichen Funktionsumfang und daher sind die Befehle unterschiedlich.
Wenn Sie das Programm gestartet haben erhalten Sie folgende Anzeige.



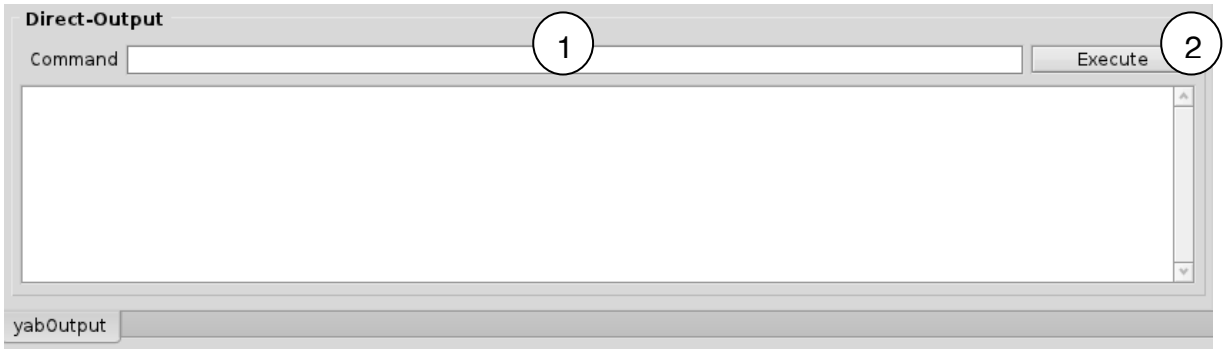
1. Im linken Teil des Feldes werden Sie, wenn Sie eine Datei geladen oder eine Datei gespeichert haben, den Dateinamen inkl. des Pfades sehen. Der rechte weiße Balken wird sich, wenn Sie eine Datei laden, langsam füllen und je nach Größe der Datei sichtbar in der Farbe verändern. Wenn dieser Grün ist, ist die Datei vollständig geladen.

2. Der Bereich Direct-Output ermöglicht Ihnen Befehle direkt zu testen, ohne sie in einem Programm zu integrieren. Ich selbst habe schon eine komplette for Schleife getestet und es ging ohne Probleme. Der Ausgabebereich (siehe 2.1) befindet sich direkt darunter. Eine Weiterführende Erklärung finden Sie unter Kaptiel 4.1.
3. Wenn Sie ein Programm innerhalb des Yab Concept Creator starten werden ihnen in diesem Ausgabebereich alle Fehler oder ihre Ausgabe der Print Befehle angezeigt. Eine Weiterführende Erklärung finden Sie unter Kaptiel 4.2.
4. In diesem Bereich haben Sie verschiedene Toolbars. Diese Toolbars können Sie nach ihrem Belieben anpassen. Eine Weiterführende Erklärung finden Sie unter Kaptiel 4.3.
5. Der Bereich mit der 5 beinhaltet mehrere Punkte. In der Grundansicht, wie Sie sie hier sehen, ist der Tab Project aktiv. In diesem werden Sie ihre gespeicherten oder geladenen Dateien finden. Die Detaillierte Erklärung finden Sie im Kapitel 4.4.

4.3 Direct-Output

Möchten Sie in dem YABConceptCreator Befehle direkt ausführen, so haben Sie die Möglichkeit in dem Reiter yabOutput unter Direct-Output in der Command ihre Befehle einzutragen und mit Execute auszuführen. Für den Direct Output dient folgendenes Beispiel.

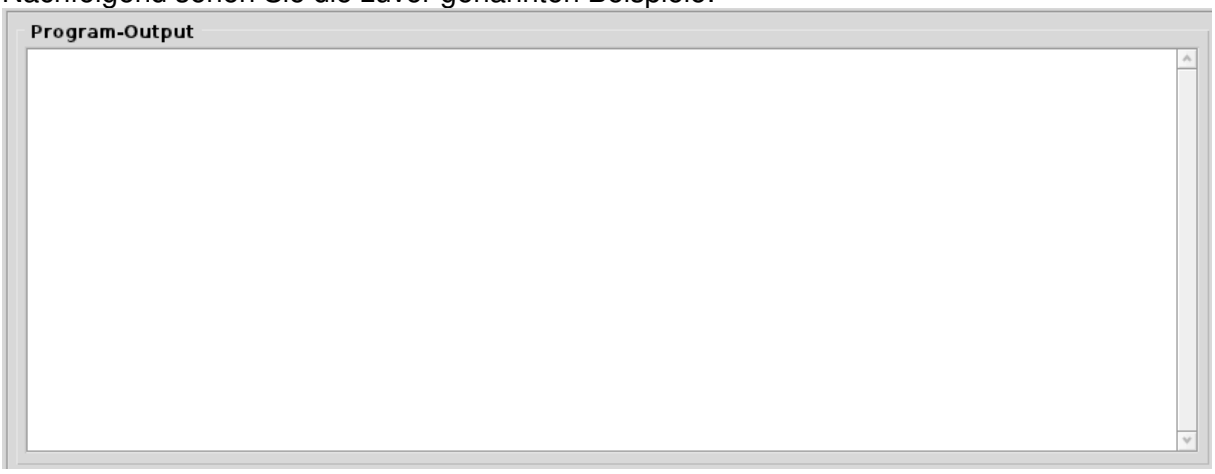
for i=1 to 10: print i: next i



Tragen Sie das die obige Befehlszeile in die Command Zeile (1) ein und klicken Sie dann mit der LMT auf Execute (2). Sie erhalten dann folgende Ausgabe.

4.4 Program-Output

Wenn Sie Programme ausführen und Sie haben print Befehle enthalten, so werden diese Ergebnisse in dem Programm-Output Bereich innerhalb des yabOutput Reiters dargestellt. Sollten das Programm aus irgendeinem Grund einen Fehler ausweisen, so erhalten Sie auch hier die Ausgabe. Auch wenn Sie das Programm später builden (sprich, als Standalone Version generieren) bekommen Sie auch hier wieder die Ausgabe über dem Erfolg oder Misserfolg.
Nachfolgend sehen Sie die zuvor genannten Beispiele:



Beispiel: Programmausgabe

Beispiel: Fehler beim Programmausführen

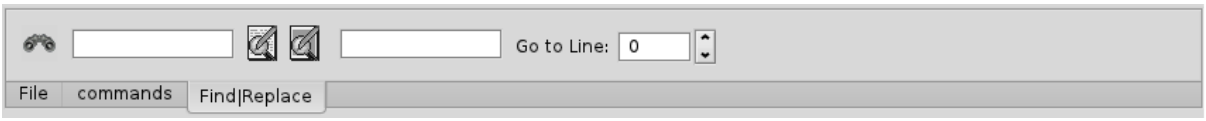
Beispiel: Programm als Standalone generieren

4.5 Toolbars

In der Toolbar Find|Replace können Sie nach Wörter oder Zahlen innerhalb ihres Quelltextes suchen. Sie haben weiterhin die Möglichkeit auch Wörter oder Zahlen zu ersetzen. Zu guter Letzt steht Ihnen noch die Möglichkeit bereit, anhand von Zeilennummer im Quelltext zu springen.

4.5.1 Suchen

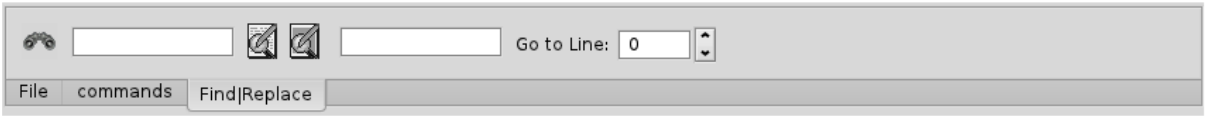
Tragen Sie im linkem Textfeld das Suchwort ein und drücken Sie links auf das Fernglas. Sie springen dann innerhalb ihres Quelltextes zu dem ersten gefundenen Suchbegeriff. Wenn Sie den Suchbegriff weiter suschen wollen, da dies nicht die gewünschte stelle war, so können Sie wieder auf das Fernglas drücken, Sie finden die nächste Stelle im Quelltext.



Für den Fall das Sie lieber mit Zeilennummer arbeiten, gibt es die Möglichkeit das Sie im rechten Textfeld (Go to Line) ihre Zeilennummer eintragen und Enter drücken. Sie können auch die Zeilennummer über die Pfeiltasten definieren.

4.5.2 Ersetzen

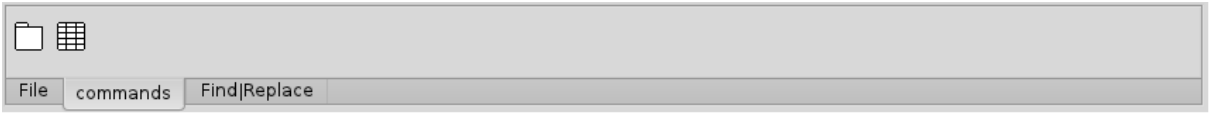
Sie haben in ihrem Quelltext einen Tippfehler bemerkt und möchten diesen recht einfach wieder korrigieren. Dann nutzen sie doch einfach die Suchen und ersetzen Funktion innerhalb des YABCC's. Diese finden Sie genauso wir die Suchen Funktion in dem Reiter Find|Replace.

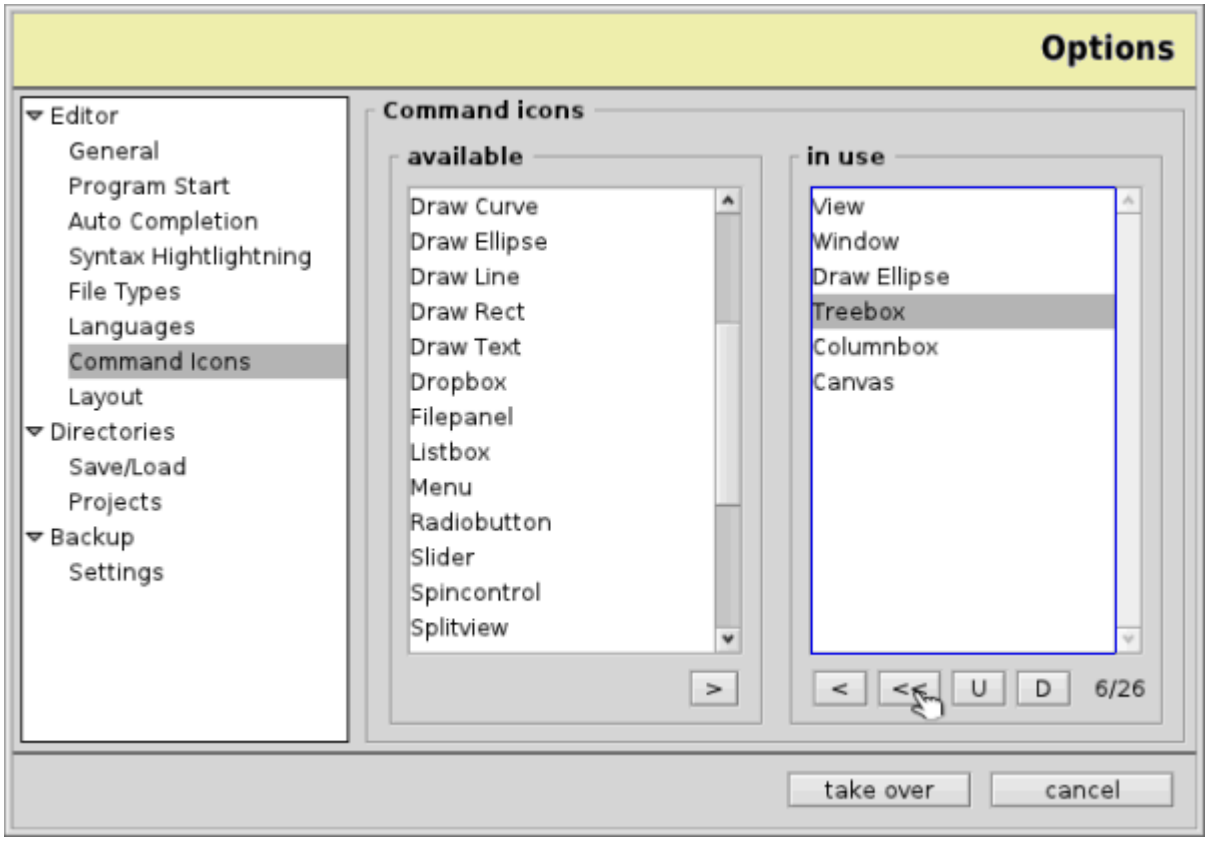


Um einen Begriff innerhalb ihres Quelltextes auszutauschen, tragen Sie bitte im leinken Textfeld den Suchbegriff und im rechten Textfeld daneben den richtigen Text der den Suchbegriff ersetzen soll ein.
Sie haben dann die Möglichkeit einzelnt den Quelltext mit suchen und ersetzen zu bearbeiten, oder alles auf einmal. Für den ersten Fall drücken sie auf das linke Icon zwischen den Textfeldern. Wenn Sie weitere Ersetzungen vornehmen wollen, dann drücken Sie wieder auf das Icon.
Möchten Sie alles auf einmal ersetzen dann drücken Sie auf das rechte Icon zwischen den beiden Textfeldern.

4.5.3 Commands

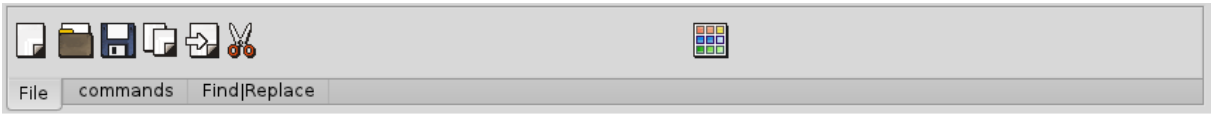
Die Toolbar commands läßt sich nach belieben konfiguieren. Sie können ingesamt 26 Icons platzieren. Dazu nutzen Sie bitte die Einstellmöglichkeit unter Options->Editor>Command Icons. Sie sehen rechts in dem Bild die bisher eingestellten Icons. Diese werden auch so wie Sie da sind in der Reihenfolge dargestellt. Wenn Sie die Reihenfolge ändern möchten so wählen Sie bitte ein Eintrag aus und bewegen diesen mit den Icons U (UP/Aufwärts) und / oder D (Down/Abwärts) nach oben bzw. nach unten. Möchten Sie weitere Icons hinzufügen bzw. aus der bisherigen Liste löschen, so nutzen Sie bitte das Icon mit dem Pfeil nach rechts zum hinzufügen und die Pfeiltaste nach links zum Löschen. Die Doppelpfeiltaste nach links, bedeutet, das Sie die ganze Liste löschen.





Wenn Sie ihre Einstellungen vorgenommen haben, nutzen Sie bitte den takeover Button zum speichern, damit ihre Änderungen auch übernommen werden.

4.5.4 File

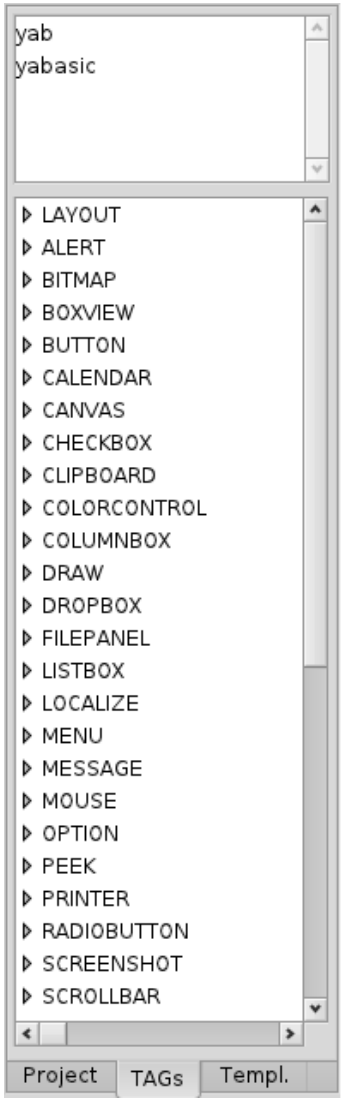


Unter der Toolbar File finden Sie von links nach rechts folgende Icons.

New	für ein neues Programm
Open	Öffnen eines bestehendes Programms
Save	zum speichern des geöffneten Programms
Copy	zum kopieren des markierten Textes
Paste	zum einfügen des markierten Textes
Cut	zum auschneiden des markierten Textes

4.6 Project, Tags und Templates

Sie können in dem YABConceptCreator ihre eigenen Projekte verwalten. Dazu nutzen Sie den Reiter Project. Desweiteren haben Sie auch die Möglichkeit, ganz schnell Befehle nachzuschlagen und gegebenfalls durch einen doppelklick in ihrem Quellcode einzubinden.
Sie können auch von uns vordefinierte Templates verweden, bzw. eigene Erstellen und Verwalten. Die den nachfolgenden Kaptieln werden ihnen diese Funktionen erläutert.



4.6.1 Project

Sie können in dem YABConceptCreator ihre eigenen Projekte verwalten. Dazu nutzen Sie den Reiter Project.

4.6.2 Tags

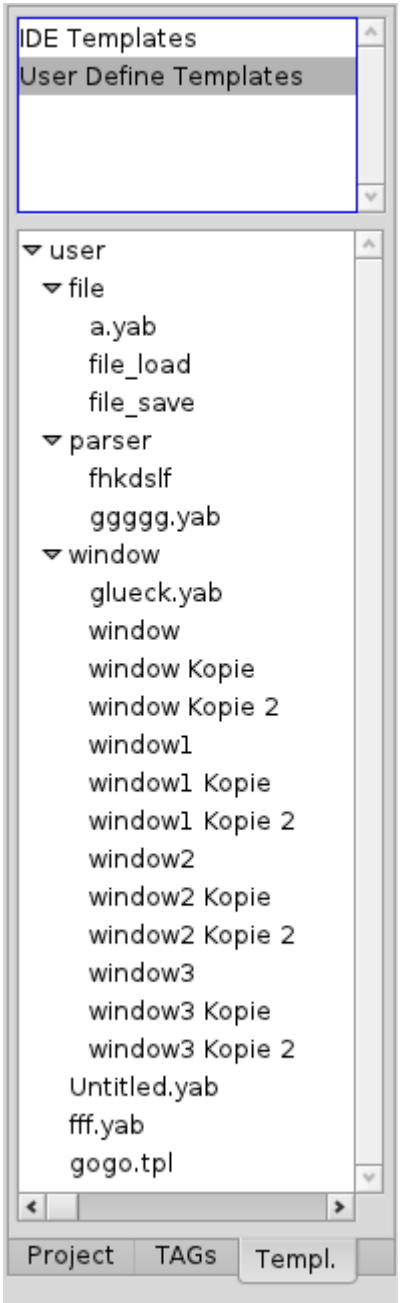
Hinter dem Begriff TAGs befinden sich alle Befehle mit Erklärungen die es in YAB gibt. Die Befehle sind unterteilt zwischen Yabasic und YAB. Das resultiert daraus, das Yabasic der Ursprung von YAB war und ist. Die grafischen und weitere Befehle sind in YAB abgelegt. Ich bin der Meinung, dass man YAB mittlerweile als Eigenständig betiteln sollte, da so vieles hinzugekommen und vieles entsprechend an Haiku angepasst worden ist, das nur noch ein paar Grundbefehle identisch sind.

Wenn Sie den Reiter TAGs das Erste mal aufmachen, werden Sie gleich die Anzeige für die grafischen Befehle in YAB sehen. Diese sind in einem Baummenu zusammen gefasst. Unter dem jeweiligen Hauptpunkt finden Sie alle dazugehörigen Befehle. Wenn Sie dann einen Befehl anklicken (LMT), dann wird ihnen rechts daneben in der Anzeige YabOutput der entsprechende Befehl mit Erklärung (eventuell sehen Sie auch ein oder mehrere Beispiel/e).

Wenn Sie ein Doppelklick auf diesen Befehl machen, wird dieser in dem zuletzt geöffneten Dokumenten-TAB eingefügt. Sollte kein Dokument geöffnet sein, wird auch nichts passieren.

4.6.3 Templates

Wir haben in dem YABConcept Creator die Möglichkeit geschaffen, ihnen Templates anzubieten, bzw. Sie können selbst Templates anlegen und verwalten. Die YABConceptCreator Templates liegen in einem vordefinierten Verzeichnis und können bzw. sollten nicht verändert werden. Sollten neue Templates von uns hinzukommen, können Sie diese entweder von der Webseite runterladen oder die Updatefunktion in den Einstellungen aktivieren.



Die von uns erstellten Templates finden Sie in dem TAB Templ. unter IDE-Templates. Diese sind genauso wie Sie rechts sehen in Kategorien unterteilt.

Möchten Sie eingene Templates anlegen, so klicken Sie bitte erstmal auf User Define Templates. Dann legen Sie einen neuen Ordner an und speichern ihr Template in diesem Ordner.

Möchten Sie das ein Template von Ihnen mit in dem YAB Concept Creator ausgeliefert wird, nutzen Sie uns bitte das Formular auf der Webseite ycc.besly.de. In diesem Formular laden Sie das Template hoch und akzeptieren das es unter GNU Lizenz weiterverwendet werden darf. Wir werden an ihrem Template nicht verändern. Deshalb sollten Sie einige Regel bei der Templateveröffentlichung beachten.

1. Das Template muss die E-mail Adresse enthalten (als REM eingetragen), ansonsten wird das Template nicht veröffentlicht. Den Support für dieses Template müssen vornehmen.
2. Die Checkbox aktvieren für die Lizenzbedinung.

5. Erste Schritte in YAB

Sie sollten sie gleich von vornherein angewöhnen ihren Quellcode einzurücken und mit Kommentaren zu versehen. Dies erleichtert ihnen das suchen nach bestimmten Codepassagen, sowie dient es der Lesbarkeit ihres Quelltextes. Auch wenn Sie jetzt sagen „der Editor hat doch eine Funktion dafür“, diese wird aber meistens nicht ihren persönlichen Maßstäben gerecht. Kommentare können sie einfach einfügen. Sie brauchen am Ende der Zeile nur ein Leerzeichen gefolgt von 2 Schrägstrichen eingeben, dahinter können Sie ihren Kommentar schreiben.

Print “Dies ist ein Text“ //Das ist mein Kommentar

Desweiteren haben Sie die möglichkeit Remark zu schreiben. Diese Remark dienen dazu, komplexere Programmabschnitte oder Informationen zum Programmierer, Datum, Version, etc. zu beschreiben.

```
REM *** Programmiert von
REM *** Datum
REM *** Version
```

Es gibt noch den DOC Befehl. Dieser ist für interne Lesezeichen (Boomarks) gedacht. Mit diesen Befehl haben Sie in der BeSly-IDE die Möglichkeit durch eine Dropbox an die gewünschte Stelle im Quelltext zu springen.

```
DOC MausMessage
```

Das nachfolgende Programm dient zur Verdeutlichungen der zuvorgenannten Punkte.

```
REM **** Program 1 ****
REM **** Author: xyz ****
REM **** Version ****
REM **** Date ****

REM **** Count 100 to 1 ****

for i = 100 to 1                    // Zählschleife von 100 auf 1
        if (i=100) then             // Bei 100 soll Start ausgeben werden
                print „Start“
        elseIf (i=50) then          // Bei 50 soll Start ausgeben werden
                print „Halftime“
        elseIf (i=0) then           // Bei 0 soll Start ausgeben werden
                print „End“
        endif
wait 0.5
next i
```

5.1 Fehler überprüfen (offen)

Folgende und andere Fehler können während des Programmierens auftreten.

- Sie definieren eine Variable doppelt und erzeugen dadurch ein Fehlverhalten des Programms.
  - ✓ **Lösung:** sprechende Namen für ihre Variablen und / oder in Subroutinen die Variablen auf Local setzen.

- Fehlende Klammer bei Schleifen und Bedingungen.
  - ✓ **Lösung:** Programm starten und die Fehlermeldung im Editor prüfen. Eventuell auch ober und unterhalb der angegebenen Zeile die Programmzeilen prüfen, da es sein kann das der Fehler durch eine andere Zeile erzeugt wurde.

- Abfragen funktionieren nicht so wie gewollt.
  - ✓ **Lösung:** Print Befehl möglichst vor der Abfrage einsetzen und die Entsprechende Variable ausgeben, manchmal kann es hilfreich sein, auch auf nicht sichtbare Leerzeichen zu prüfen. Auch bei der Abfrage selbst kann sich auch ein Leerzeichen beim tippen des Quelltextes einschleichen.

  Beispiel1:

```
text$="h "
Print text$

If (text$="h") then
        Print „stimmt“
endif
```

  Beispiel2:

```
text$="h"
Print text$
If (text$="h ") then
        Print „stimmt“
Endif
```

- Array können nicht definiert werden oder vergrößern sich nicht wie gewollt
  - ✓ **Lösung:** Überprüfung der Definition des Array und / oder Überprüfung der Wertzuweisung in das Arrays.

Weitere Fehler können natürlich auftreten. Diese kann ich nicht kennen, aber Sie haben die möglichkeit auf der Seite beusergroup.de in dem Forum nach erfolgreicher Anmeldung unter Entwicklung -> YAB fragen zu stellen. Es wäre Hilfreich, wenn sie erst die Suche verwenden, bzw. selbst in dem Forum suchen, ob jemand dieses Problem schon hatte. Dadurch können doppeltpost vermieden werden.

Desweiteren steht ihnen unter besly.de eine YAB Wissenbasis zur Verfügung. In dieser Wissensbasis finden Sie hilfreiche Teilprogramme, Erklärungen und Lösungen zu vielen Problemstellungen.

6. Variablen, Arrays und Konstanten

Stellt sich Ihnen jetzt die Frage was Variablen und Konstanten sind und wofür Sie Diese jetzt benötigen. Eine Variable benötigen Sie dafür Informationen zu speichern, solange bis Sie diese wieder benötigen oder mit einem neuem Wert überschreiben.

Konstanten dagegen sind sozusagen Schreibgeschütze Variablen. Diese können sie nur lesen und nicht verändern. Diese können zum Beispiel der Wert PI sein oder auch von ihnen vordefinierte Zeichenketten `\n` (new line)

In Yab gibt es nur 2 Variablen Typen. Grundsätzlich gilt für beide Variablentypen dass der Variablenname immer mit einem Buchstaben anfangen muss und keine **Umlaute** sowie **ß** beinhalten darf. Des Weiteren sollten Sie auch immer sprechende Namen verwenden, wie z.B. EingabeTastatur oder zaehler.
Nur x oder a zu schreiben erschwert ihnen bei späteren Änderungen oder Kontrollieren des Programms den Fehler oder die Stelle wieder zu finden, bzw. Sie können nicht mehr schnell feststellen woher der Wert kommt.

Kommen wir jetzt wieder zurück zu den Variablentypen String und Numero. Bei der Numerovariable wird nur der Name der Variable geschrieben, z.B. Zaehler, und bei der Stringvariable wird der Name gefolgt von einem Dollarzeichen **$** geschrieben, z.B. EingabeTastatur$. Numero und Text-Variablen können nur einen Wert speichern. Möchten Sie einer dieser Variablen mehrere Werte zuweisen, so müssen Sie Arrays verwenden. Des Weiteren können Sie bei einer Numerovariablen nur Zahlen speichern, wenn Sie versuchen einen Buchstaben in eine Zahlvariable zu speichern, so müssen sie dieses Zeichen erst konvertieren.

Was Sie auch noch beachten sollten, so gibt es einige Reservierte Wörter die Sie als Variable nicht verwenden dürfen, genauso darf eine Variable nicht genauso heißen, wie ein Befehl. Eine Auflistung der Reservierten Wörter finden Sie im Anhang.

Nachfolgend sehen jeweils ein Beispiel wie Sie einen Wert einer Numero, bzw. einer String Variablen zuweisen können.

Beispiel Numero:

```
x=10
```

Beispiel Addition von zweier Numerovariablen:

```
x=10
y=10
z=x+y
```

Bespiel String:

```
text$="Dies ist meine erste Textvariable"
```

Beispiel Verkettungen von Stringvariablen:

```
Text2$="und dies ist meine zweite Textvariable"
text$="Dies ist meine erste Textvariable"+text2$+"Ende"
```

Beispiel Verkettungen von Stringvariablen und Numero:

```
x=10
Text2$="und dies ist meine zweite Textvariable"
text$="Dies ist meine erste Textvariable"+text2$+"Ende"+str$(x)
```

Diese Beispiele sollten ausreichend sein, um ihnen die Zuweisung von Werten zu einer Variablen zu verdeutlichen.

6.1 Arrays

Um mehrere Werte in einer Variablen zu speichern, müssen Sie Arrays verwenden.
Arrays können so wie Variabeln Numero und String Inhalte speichern. Dazu müssen Sie genauso das Dollarzeichen $ für Stringarrays verwenden, wie bei den Variablen auch. Arrays müssen vor ihrem ersten Gebrauch deklariert werden, damit YAB weiß wieviel Speicher ersteinaml reserviert werden muss. Zum Glück wird der Speicherverbrauch auch automatisch vergrößert, wenn dies von nöten ist.

6.1.1 Eindimensionales Array

Ein einfaches Array mit nur einer Dimension nennt man eindimensionales Array. Dimension steht für einen Index der durch eine Zahl defniert wird.

Ein eindimensionales Array erstellen Sie mit
DIM Arraynamen(Dimension) // für Numero-Arrays
DIM Arraynamen$(Dimension) // für String-Arrays

6.1.2 Mehrdimensionales Array

Sie können auch Arrays mit mehrere Dimensionen erstellen, diese werden dann mehrdimensionales Arrays genannt. Die Maximale Anzahl von den Dimensionen ist auf 10 begrenzt, d.h. Sie könnten, wenn Sie es bräuchten, folgendes Array erstellen.

DIM Arraynamen$(10,20,10,10,80,10,10,10,40,10)

Redim

6.1.3 Diverse Größen von Array's ermitteln

arraysize
arraydim

6.2 Konstanten

Konstante benötigen Sie immer dazu, wenn Sie immer wiederkehrende Werte verwenden wollen. Leider kann man in YAB keine eigenen Konstanten definieren, sondern man nur die vordefinierten verwenden.

Diese vordefinierten Werte finden Sie in der nachfolgenden Tabelle und in dem Kapitel 101.

Konstantennamen	Wert		Konstantennamen	Wert
pi (oder PI)	3.14159			
euler	2.71828182864			

7. Mit Zahlen arbeiten

Sie können in Yab mit Zahlen rechnen. Das bedeutet das sie auf jeden Fall die Grundrechenarten, sowie komplexere Berechnungen, durchführen können.

Eine Übersicht der ihnen zur Verfügung stehenden Operatoren.

+	Addition	
-	Subtraktion	
*	Multiplikation	
/	Division	
^ oder **	Potenzieren	siehe 6.1.8 und 6.1.9
y**x/(1/n-te Wurzel)	Wurzel	siehe 6.1.8 und 6.1.9

```
print 1+2        ergibt 3
print 2*3        ergibt 6
print 4/2        ergibt 2
print 2^3        ergibt 8
print 2^3(1/3)   ergibt 2
```

Sie haben auch die Möglichkeit Winkelberechnungen anzustellen, oder Bruchrechen-aufgaben, die Quadratwurzel berechen zu lassen. Die dazugehörigen Befehle lesen Sie im Abschnitt 7.1 Rechen-funktionen.

7.1 Rechenfunktionen

7.1.1 Trigonometrische Funktionen

Es gibt 6 trigonometrische Funktionen:

```
print sin(1.0),cos(pi),tan(3)
print asin(0.5),acos(0.7)
print atan(2),atan(1,2)
```

Diese Zeilen erzeugen diese Ausgaben:

```
0.841471 -1 -0.142547
0.523599 0.795399
1.10715 0.463648
```

Wie sie sehen, kann Yabasic mit Sinus (sin), Kosinus (cos), Tangens (tan) und ihren Kehrwerten rechnen. Alles diese Funktionen erwarten die Werte im Bogenmaß.

Um die Umrechnung von Grad in Bogenmaß zu ermöglichen (Bogenmaß=Grad*pi/180), gibt es die reservierte Variable pi (oder PI), mit dem festen Wert 3.14159.

Beachten Sie bitte auch, dass die atan() Funktion in zwei Arten erscheint.
Mit einem Argument (z.B. atan(2)) atan() ergeben sich ein Werte zwischen
pi/2 ... +pi/2.

Aufgerufen mit zwei Argumenten (z.B. atan(2,-1)) atan() ergeben sich Werte zwischen -pi und +pi.

(Dies kann hilfreich sein z.B. zur Transformation vom kartesischen in das polare Koordinatensystem).

Die atan Funktion (Arcustangens-Funktion)ist eine Invertierte tanges-Funktion. Diese gibt den Winkel im Bogenmaß, nicht Grad! zurück.
Die auf die tan-Funktion zugeführt wird die übergebene Argument der atan-Funktion zu erzeugen.

Die atan-Funktion akzeptiert auch zwei Argumente. Diese gibt dann einen Winkel im Bereich -π bis π zurück. Wahrend dessen die 1 Agument Variante einen Bereich von -π / 2 bis π / 2 zurückgibt.
Da der einfache Arkustangens nicht die Möglichkeit bietet, den Winkel im korrekten Quadranten zu ermitteln, und außerdem die Tangensfunktion für einen Funktionswert von nicht umkehrbar ist, gibt es in vielen Programmiersprachen eine Funktion, die mit 2 Argumenten aufgerufen wird.

Example

```
print atan(1),atan(tan(pi)),atan(-0,-1),atan(-0,1)
```

This will print 0.463648 2.06823e-13 -3.14159 3.14159 which is π/4, almost 0, -π and π respectively.

7.1.2 Integer-/Ganzzahlen und Bruchteile

Die Funktionen int() und frac() zerlegen das Argument am Dezimalpunkt.
Wenn Sie die Funktion int benutzen, bekommen Sie nur die Ganze Zahl vor dem Dezimalpunkt zurück. Wenn Sie die Funktion frac benutzen bekommen Sie die Zahl hinter dem Dezimalpunkt zurück.

```
print int(11.8) ergibt: 11
print int(-11.8) ergibt: -11

print frac(11.3147) ergibt: 0.3147
print frac(-11.3147) ergibt: 0.3147
```

7.1.3 Absolute Werte

Die abs() gibt den absoluten Wert (ohne Vorzeichen) des Arguments wieder:

```
print abs(-2.34)    ergibt 2.34
print abs(2.34)     ergibt 2.34
```

7.1.4 Das Vorzeichen einer Zahl bestimmen

Die sig() Funktion gibt einen Wert in Abhängigkeit des Vorzeichens des Arguments wieder:

```
print sig(-2.34)  ergibt -1
print sig(0)          ergibt 0
print sig(2.34)     ergibt: +1
```

Diese Funktion kann man gut Zahlen automatisch farbig in Abhängigkeit ihrer Wertigkeit darzustellen.

7.1.5 Rest einer Division

Um den Rest einer Division zu erhalten, ist die Modula-Funktion mod() anzuwenden. z.B. mod(11,4) erzeugt 2. Sie bekommen nur den Restwert zurück.
Die Division von 11 durch 4 ergibt 2, da 2 *4=8 ist, und Sie haben einen Rest von 3 bis zur 11.

7.1.6 Minimum und Maximum

Zeigt den niedrigsten bzw. den höchsten Wert von zwei Argumenten an.

```
print min(2,3)      ergibt 2
print max(2,3)      ergibt 3
```

7.1.7 Quadratwurzel und Quadratzahl

Die Quadratwurzel wird berechnet mit sqrt() und die Quadratzahl mit sqr().

```
print sqrt(2)       ergibt 1.41421 4
print sqr(2)        ergibt 4
```

7.1.8 Potenz und Wurzel

Um eine Potenz einer Zahl zu berechnen, könnten Sie einen ganz lange Zeile schreiben, wo sich die Zahl mit sich selbst immer wieder multipliziert, z.b. für eine Potenz 2 hoch 4.

```
print 2*2*2*2     ergibt 16
```

Diesen Zustand möchte man ja niemanden zumuten und der Entwickler von Yabasic hat sich folgendes ausgedacht. Man schreibt jetzt nur noch print 2**4 oder 2^4. Die erste Zahl ist die Basis, also die Zahl die Potenziert werden soll, die Zweite Zahl in diesem Beispiel ist die 4 der Exponent.

```
Print 2**4        ergibt 16
Print 2^4         ergibt 16
```

Um eine dritte oder n-te Wurzel zu berechnen reicht der Befehl sqrt() nicht aus, da dieser nur die Quadratwurzel berechnet. Also müssen Sie sich jetzt anders behelfen.
Dazu haben Sie die Möglichkeit die Potenzierung umzukehren. Die Schreibweise lautet dann a**b/(1/wurzel). Sie potenzieren erst und dann teilen Sie es durch die gewünschte Wurzel. Also, wenn die Variable wurzel = 4 ist, wurden Sie die 4te Wurzel ziehen. In dem nachfolgenden Beispiel potenzieren wir erst eine Zahl und ziehen dann die entsprechende Wurzel um auf den Ursprung der Basis zurückzukommen. Wobei wir es ja eigentlich wissen.

```
print 4**3                ergibt 64
print 4**3/(1/3)  ergibt 4
```

Sie können natürlich auch die zweite Schreibweise mit dem Dach verwenden.

7.1.9 Exponenten und Logarithmus

Die Expontentailfunktion und die Logarithmusfunktion arbeiten beide mit der Basis Euler, wobei bei log auch eine andere Basis angegeben werden kann.
Euler ist eine Konstante (2.71828), die nach Leonard Euler benannt wurde.

```
print exp(1)      ergibt 2.71828
```

Die exp(n)-Funktion nimmt sich n-mal selbst mal, d.h. wenn Sie exp(3) schreiben, erhalten Sie als Ergebnis 20.0855369. Sie könnten auch schreiben.

```
print euler*euler*euler    ergibt 20.0855369
```

Die log() Funktion gibt den Logarithmus einer Zahl zu der angegebenen Basis zurück.
Wenn Sie bei log() die Basis weglassen, wird automatisch die Konstante Euler verwendet.

```
print log(x, basis)

print log(2)      ergibt 0,69314718
print log(8,2)    ergibt 3 (Das ist der Logarithmus von 8 zur Basis 2 (3))
```

7.1.10 Umwandlung von Zahlen

Um eine Dezimalzahl in eine Hexdezimalzahl umzuwandeln und umgekehrt, benutzt man hex$() und dec():

```
print hex$(255)          ergibt ff
print dec("ff")          ergibt 255
```

Ganz ähnlich funktioniert die Umwandlung in binäre Zahlen: So ergibt bin$(8) den Wert "100" und umgekehrt hex("100",2) den Wert 8; dabei verlangt die funktion dec() als zweites Argument die Basis (hier: 2), wenn sie vom Standardwert 16 abweicht.

7.1.11 Zufallszahlen

Zufallszahlen werden erzeugt mit der ran()-Funktion. Diese Funktion kennt zwei Arten: Aufgerufen ohne Argument (z.B. print ran()) erhält man eine Zufallszahl zwischen 0 und 1. Aufgerufen mit einem Argument (z.B. print ran(2)) erhält man eine Zufallszahl zwischen 0 und dem vorgegebenen Argument.
Die ran()-Funktion von Yabasic benutzt die ran()-Funktion der C standard library, deshalb darf man nicht ein zu gutes Ergebnis an Zufallszahlen erwarten.

```
print ran()
print ran(2)
```

7.1.12 Bitmanipulation (offen)

Sie können die bitweisen Operationen and(), or(), eor() (der auch als xor() geschrieben werden kann), benutzen wie z.B. hier:

```
print and(10,7)  ergibt 2
print or(9,3)            ergibt 11
print eor(15,4)          ergibt 11
```

Achtung!

Beachten Sie bitte, das Yab keine not() Operation kennt.

Die Wirkung der Operatoren soll an einem Beispiel erläutert werden:

```
int i = 2, n = 1;
int m;

m = i & n;          /* liefert m = 0
                    ein Bit von m ist nur dann auf 1 gesetzt,
                    wenn die entsprechenden Bits von i und n
                    gleichzeitig auf 1 gesetzt sind */
m = i | n;          /* liefert m = 3
                    ein Bit von m ist dann auf 1 gesetzt,
                    wenn mindestens eines der entsprechenden
                    Bits von i und n auf 1 gesetzt ist */
m = i ^ n;          /* liefert m = 3
                    ein Bit von m ist dann auf 1 gesetzt,
                    wenn die entsprechenden Bits von i und n
                    verschiedene Werte besitzen */
m = i << n;         /* liefert m = 4
```

```
                entspricht einer n-maligen Multiplikation
                von i mit 2 */
 m = i >> n;      /* liefert m = 1
                entspricht einer n-maligen Division
                von i durch 2 */
 m = ~i;         /* liefert m = -3
                jedes Bit von m wird aus dem Komplement
                des entsprechenden Bits aus i gebildet */
Beispiel:
 int i, n;

 i = 1 << n;       /* berechnet die n-te Potenz von 2 */
```

8. Strings manipulieren

Nachdem wir jetzt wissen, wie man mit Zahlen arbeiten kann, wollen wir jetzt erforschen wie man Strings manipulieren kann. Dazu gehört das ab- / aus- schneiden von Stringteilen oder das groß / klein schreiben von Strings. Es gibt auch die Möglichkeit Zahlen in einen String umzuwandeln und umgekehrt, allerdings dies dann mit Einschränkungen.

8.1 Einen String in Teilstücke zerlegen

Um einen String zu zerlegen, haben Sie die Möglichkeit zwischen 3 Befehlen zu wählen. Es kommt ganz darauf an, was Sie machen möchten.

1. Der Befehl **left$()** schneidet so viele Buchstaben von links ab, wie Sie es im zweiten Argument angegeben haben.

   text$ = left$(“text“, 2) //schneidet die linken beiden Buchstaben aus dem Wort text aus
   print text$

   ergibt **te**

2. Der Befehl **right$()** schneidet den Teil rechts vom String ab.

   text$ = right$(“text“, 2) //schneidet die rechten beiden Buchstaben aus dem Wort text aus
   print text$

   ergibt **xt**

3. Der Befehl **mid$()** schneidet aus der Mitte aus, wobei das zweite Argument die Startstelle und das dritte Argument die Anzahl der auszuschneidenden Zeichen angibt. Wenn Sie das dritte Argument weg lassen erhält man den String von der angegebenen Position bis zum Ende:

   mid$("Hallo",2) // gibt also "allo" zurück.

Darüber hinaus kann man left$(), **mid$() und right$()** dazu benutzen, um ausgewählte Teile eines Strings zu ersetzen.

a$="123456"
left$(a$,2)="abcd"
print a$
ergibt **ab3456**

Wie Sie sehen wurden nur die beiden linken Buchstaben ausgetauscht (obwohl der String "abcd" vier Buchstaben enthält). Dasselbe können Sie mit mid$() oder right$() erreichen. Beachten Sie, dass sich dadurch nicht die Länge des Ursprungsstrings ändert, sondern nur der Teil geändert, bzw. ausgeschnitten wird.

Sie haben Sich schon bestimmt überlegt, wie aufwendig es sein wird, einen längeren String in einzelne Teile zu zerlegen.

8.1.1 Zerlegen eines Strings in Teilstrings (Token)

Es gibt zwei Funktionen um einen String in Teilstrings (Token) zu zerlegen. Die Funktionen erwarten folgende Parameter. Einen String, der in Teilstrings (Token) zerlegt werden soll und ein Array, in dem die Zeichen abgelegt werden. Bei dem Befehl Token können Sie auch noch ein drittes Argument angeben. Dieses dritte Argument ist dafür da, dass der String an dem angegebenen Zeichen zerlegt wird. Es können bei diesem Argument auch mehrere Zeichen hintereinander angegeben werden. Dann wird entsprechend dem gefundenen Zeichen der String zerlegt, z.B. **token(a$,words$(),":;")** .

token() achtet auf die erzeugten Zeichen und überspringt ein oder mehrere Trennzeichen, wenn sie direkt aufeinander folgen. **token()** gibt keine leeren Zeichen zurück.

Ein Beispiel:

```
ausganztext$=" eins zwei drei "

dim words$(1)

num=token(ausganztext$,words$()," ")

for a=1 to num
      print words$(a)
next a
```

Das Programm erzeugt folgende Ausgabe:

```
eins
zwei
drei
```

8.1.2 Zerlegen eines Strings in Teilstrings (Split)

Wie wir sehen wird das Array **words$()** nach Bedarf vergrößert. Der Inhalt des Arrays wird jetzt mit den neuem Überschrieben. Beim Übergeben der Werte an das Array dürfen sie keine Indizes übergeben, z.B. **a$(2)** ist falsch, **a$()** dagegen richtig.

Die Zweite Funktion zum Zerlegen eines String ist die Funktion split(). Die Funktion **split()** achtet auf das Trennzeichen. Wenn jetzt der zu zerlegende String x Trennzeichen enthält, wird die Funktion die Zeichen zwischen den Trennzeichen zurückgeben. Wenn da keine Zeichen zwischen den Trennzeichen zur Verfügung stehen wird Split ein leeres Zeichen zurückgeben.

```
l$="::eins::zwei:drei::vier"
dim words$(1)
num=split(l$,words$(),":")

for a=1 to num
      print "Token: "+words$(a)
next a
```

Das Programm wird diese Ausgabe erzeugen:

```
Token:
Token:
Token: eins
Token:
Token: zwei
Token: drei
Token:
Token: vier
```

8.2 Strings (Text) in Zahlen verwandeln und umgekehrt

Die Funktion **str$()** wandelt das numerische Argument in einen String um:

print str$(12) ergibt den String (Text) "**12**" als Resultat.

Die Formatierung der Zahl kann durch ein wahlweise weiteres Argument bestimmt werden:
print str$(12.123455,"##.##") ergibt den String **12.12**.

Das zweite Argument hat denselben Effekt wie der Formatstring des print using Befehls.

Genau das Umgekehrte macht die Funktion **val()**:

print 2+val("23") ergibt **25** als Resultat, während **print val("e2")** eine **0** liefert, weil **"e2"** keine gültige Zahl ist.

8.2.1 Zeichensätze

Yab bietet zwei Funktionen an, um mit einem Zeichensatz zu arbeiten.

1. Die erste Funktion ist **asc()**. Diese gibt die Nummer eines Zeichens im aktiven Zeichensatz zurück.

   print asc("e") gibt **101** als Ergebnis zurück, weil das Zeichen **"e"** die Nummer **101** im Zeichensatz reserviert hat.

2. Die zweite Funktion ist **chr$()**. Diese Funktion gibt das entsprechende Zeichen aus dem Zeichensatz zurück. Somit sind Sie in der Lage Umlaute oder bestimmte Symbole in ihren Text einzufügen.

Eine komplette Übersicht zu dem verwendeten Zeichensatz erzeugen sie mit dem folgenden Programm.

```
i=0
while (i<256)
  print chr$(i),"=",i,
  i=i+1
wend
```

Eine Übersicht über alle ASCII Zeichen finden Sie im Anhang.

8.2.2 Escape-Folgen

Mit **chr$()** könenn Sie auch nichtdruckbare Zeichen erstellen. Trotzdem brauchen Sie **chr$()** nicht so häufig zu verwenden wie Sie vielleicht denken. Der Grund liegt darin, dass Sie mit dem Backslash-Zeichen**(\)** Escape-Folgen bilden können.

Als Beispiel:

Print “Text\nAbstatz“

Ergibt:
Text
Absatz
Das **n** erzeugt in diesem Fall einen Zeilenvorschub (nächste Zeile), stattdessen hätten Sie auch **chr$(10)** verwenden können. In der nachfolgenden Tabelle werden ihnen alle Escape Sequenzen dargestellt.

Escape-Folgen	ASCII-Zeichen
\n	Zeilenvorschub
\t	Tabulatorsprung
\v	vertikalerTabulator
\b	Rückschritt
\r	Wagenrücklauf
\f	Seitenvorschub
\a	Glocke
\\	Backslash
\`	Einf. AnfStrich
\"	Dopp. AnfStrich

8.2.3 Wechsel zwischen Groß- und Kleinschreibung

Das Beispiel-Programm enthält einige weitere String-Funktionen:
lower$() und sein Gegenstück **upper$()** wandeln ein String-Argument jeweils in groß- oder kleingeschriebene Zeichen,
z.B. **lower$("aBcD12fG")** ergibt **"abcd12fg"**

text$=“ Dies ist ein kurzer Text mit störenden Leerzeichen am Anfang und am Ende “
print lower$(text$)
print upper$(text$)

8.2.4 Leerzeichen entfernen

Sie haben bestimmt schon mal überlegt, wie Sie eventuell störende Leerzeichen vor oder hinter einem Wert entfernen können. Man könnte jetzt umständlich die Länge des Strings ermitteln und nach Leerzeichen (also “”) suchen lassen und dann den entsprechen mit left$, right$ oder mid$ extrahieren, oder man nutzt die mitgelieferten Funktionen von YAB.
Diese wären **ltrim$()**, **rtrim$()** und **trim$()**. Sie können mit ltrim$(ihre_variable oder text) führende Leerzeichen entfernen. Mit rtrim$() können Sie nachgehende Leerzeichen aus einem String zu entfernen,
Jetzt haben Sie aber vorher und nachher Leerzeichen die Sie entfernt haben möchten. Jetzt könnten sie mit **rtrim$(ltrime$())** diese Leerzeichen entfernen oder Sie nutzen die Funktion trim$(). Diese Funktion entfernt führende oder nachfolgende Leerzeichen bei einem String.

z.B. **ltrim$(" oo ")** ergibt **"oo "** und **rtrim$(" oo ")** ergibt **" oo"**.
Und **trim$(" oo ")** ergibt **"oo"**

Beispielzeilen:

text$=“ Dies ist ein kurzer Text mit störenden Leerzeichen am Anfang und am Ende “
print **ltrim$(text$)**
print **rtrim$(text$)**
print **trim$(text$)**

8.3 Strings (Text Strings in Strings suchen)

Sie können auch nach bestimmten Zeichen oder Zeichenketten in einem Text suchen. Dazu benutzen Sie die Funktion instr(). Die Funktion instr() gibt ihnen die Position des zweiten String-Arguments innerhalb des ersten String-Arguments oder 0 wieder, falls nichts gefunden wird, oder anders gesagt, der erste Teil in den Klammern bei instr() ist der Text der durchsucht werden soll, der zweite Teil ist der Suchtext / Zeichen, das gefundenen werden soll. Sie haben noch die Möglichkeit ein drittes Argument anzugeben. Durch das dritte Argument teilen Sie der Funktion den Startpunkt der Suche mit. Als Ergebnis erhalten Sie eine Zahl zurück mit der Positionsangabe, z.B. instr("Hallo","al") gibt 2 zurück, weil "al" bei Position 2 erscheint, innerhalb von "Hallo". Aber instr("Hallo","Al") gibt 0 zurück, weil "Al" nicht in "Hallo" enthalten ist (Die Schreibweise stimmt nicht).

instr("ende","e") gibt 1 zurück,
instr("ende","e",2) dagegen 4, weil die Suche nach einem "a" ab der zweiten Stelle in "ende" erst an der vierten Stelle fündig wird.

Wenn Sie dagegen sagen, gibt es den keine Möglichkeit den String von rechts her zu durchsuchen, doch die gibt es. Sie können dafür die Funktion **rinstr()** benutzen. Diese Funktion beginnt ihre Suche von rechts her und schreitet nach links fort. Die Argument die Sie der Funktion geben müssen sind Identisch zu der Instr() Funktion. Des Weiteren haben Sie auch hier die Möglichkeit die Startposition anzugeben.

8.3.1 Globbing, Stringvergleiche mittels Platzhaltern (Wildcards)

Mit **glob()** können Sie, wie mit instr(), prüfen, ob das erste String-Argument mit dem zweiten String-Argument, dem Prüfstring, übereinstimmt. Sie erhalten dann aber als Ergebnis WAHR oder FALSCH. Das zweite Argument darf die Platzhalter **?** und ***** enthalten, wobei **?** ein beliebiges Zeichen repräsentiert und ***** für eine beliebige Anzahl beliebiger Zeichen steht.

Glob kann nur in Verzweigungen verwendet werden (z.B. nach **if**, **while** oder **until**) wie in **if (glob("Hallo","H*o")) print "Es passt !"**.

Nun einige Beispiele:

glob-Bedingung	WAHR/FALSCH ?
glob("abcd","abc?")	WAHR
glob("abab","*")	WAHR
glob("abc","ab??")	FALSCH
glob("abcdabab","ab*ab")	WAHR
glob("abcd","*a")	FALSCH

9. Wiederholen von Programmteilen

Sie können in einem Programm bestimmte Programmteile, die sie immer wieder benötigen erneut aufrufen, ohne das Sie den Code jeweils neu programmieren oder an die entsprechende Stelle kopieren. Diese erneuten Aufrufe können Sie durch Schleifen bzw. Funktionen erzeugen. Die einfachsten Schleifen sind Zählschleifen.

9.1.1 Zählschleifen

Eine Zählschleife erhöht z.B. eine Variable immer um einen bestimmten Wert.
Alles was in dieser Schleife steht wird solange wiederholt bis die Bedingung erfüllt worden ist. Es kann ihnen passieren, dass Sie unbeabsichtigt eine Endlosschleife programmieren.

```
x=0

For x to 1 to 10       // das x ist eine Variable die hochgezählt wird.
        print x
Next x
```

Diese Schleife wird 10 mal durchlaufen, da x vor der Schleife auf 0 gesetzt worden ist.

9.1.2 Geschachtelte Schleifen

Sie wollen z.B. mehrere Werte in Tabellenform ausgeben. Dazu bietet sich folgendes an. Angenommen Sie wollen das kleine Einmal Eins in tabellarischer Form ausgeben, so überlegen Sie sich erstmal wie viele Zahlen sie haben und wie Sie die Tabelle an besten aufbauen. Mein Vorschlag wäre eine Tabelle mit 10 Zeilen und 10 Spalten. Um so eine Tabelle aufzubauen benötigen Sie 2 Schleifen. Die eine Schleife ist für die Zeilen und die andere für die Spalten zuständig. Im nachfolgenden Code wird ihnen die Vorgehensweise gezeigt.

```
Zahl=0
For x to 1 to 10
        For y to 1 to 10
                zahl=zahl+1
                zahl=zahl*1
                print \%t zahl
        Next y
Next x
```

9.2 Die While Wend Schleife

Die While Wend Schleife läuft fortwährend weiter, solange bis eine Bedingung erfüllt wird. Dann wird diese Bedingung abgearbeitet und die Schleife läuft weiter. Sie müssen bedenken, das alles was vor der Schleife steht erst abgearbeitet wird und alles was nach der Schleife steht nur abgearbeitet wird, was entweder angesprochen wird, z.B. als Unterroutinen, oder nur wenn man die Schleife abgebrochen hat. Eine While Schleife kann man mit dem Befehl Break beenden.

Praktische Beispiele:

Wir nutzen diese Schleife z.B. beim auslesen von Dateien oder wenn Tastatureingaben, Maus Klicks überprüft werden sollen.

```
    fileload$="taballe.csv"
    auslesen=open(fileload$, "r")
    x=0
    while (not EOF(auslesen))
          line input #auslesen b$
          dim elements$(1)
          numElements = split(b$, elements$(),";")
          for i = 1 to numElements
              x=x+1
             dim d$(x)
             d$(x) = elements$(i)
          next i
    wend
close(auslesen)
```

Beipiel 1

```
inloop = true
while(inloop)
    msg$ = message$
    switch msg$

        Hier werden die Einträge in den Loop eingefügt.

    end switch
wend
```

Beipiel 2

In dem Ersten Beispiel wird die Datei solange Zeile für Zeile gelesen bis keine Zeile mehr gefunden wird. Danach wird die Datei wieder geschlossen und die Schleife beendet.

In dem Zweiten Beispiel läuft die Schleife fortwährend weiter, solange bis eine Bedingung in der Switch Abfrage erfüllt wird. Dann wird die Anweisung dazwischen abgearbeitet und danach, sofern das Programm nicht beendet wird, läuft die Schleife weiter.

9.3 Die Repeat Until Schleife

Die Repeat Until Schleife läuft immer weiter, bis die Bedingung in den Klammer hinter den dem Befehl Until erfüllt ist. Wie bei den anderen Schleifen zuvor wird alles was zwischen Repeat Until steht ausgeführt.

```
x=0
print "This program will print the numbers from 1 to 10"
repeat
   x=x+1
   print x
until(x=10)
```

9.4 Die Do Loop Schleife

Die Do Loop Schleife läuft immer weiter, um aus dieser Schleife herauszukommen, müssen Sie den Befehl break oder Goto verwenden. Alles was zwischen DO Loop steht wird ausgeführt.

```
do
  a=a+1
  print a
```

```
    if (a>100) break
  loop
```

Abschließend zu den Schleifen könnte man noch unterscheiden zwischen Kopfgesteuert oder Fußgesteuerte Schleifen. Bei den Kopf gesteuerten die Bedingung jeweils vor dem Ausführen der Anweisung(en) überprüft; deshalb spricht man von einer *Kopfgesteuerten* Schleife. Dazu gehört die While Wend Schleife. Die Fußgesteuerten Schleifen, wie z.B. die Repeat Until oder do loop werden auf jeden Fall mindestens einmal ausgeführt.

10. Entscheidungen Treffen

Sie kommen irgendwann in ihrem Programm an den Punkt, wo Sie Entscheidungen treffen müssen, z.B. bei einem Spiel, die Überprüfung wenn der Spieler durch die Gegend läuft und auf ein Hindernis stößt. Die ganze Zeit wird abgefragt, ob der das nächste des Spielers auf dem er sich zu bewegt frei ist. Diese Abfrage kann man z.B. mit einer IF then oder Switch Case Abfrage bewerkstelligen.

10.1 If then else (Wenn -> Dann -> Ansonsten)

Die If Then Abfrage hat viele Möglichkeiten. Sie können mit der IF Abfrage nur eine Bedingung oder mehrere Bedingungen überprüfen. Das IF steht für das Fragewort Wenn, wenn die nachfolgende in Klammern gesetzte Bedingung erfüllt wird, kommt der Befehl THEN ins Spiel. Er steht für das Wort DANN und der nachfolgende Code wird ausgeführt. Am Ende der Abfrage steht immer FI oder ENDIF (beide Schreibvarianten sind möglich. Ich habe mich an ENDIF gewöhnt und finde das die Lesbarkeit des Codes dadurch besser ist.).

Bei dem unten gezeigten Beispiel wird nur eine Bedingung überprüft.

```
if (x = 2) then          // Wenn x = 2 ist dann führe alles zwischen if und endif aus.
                         // Befehl der hier ausgeführt werden soll, wenn die
                         // Bedingung in den Klammern erfüllt worden ist.
Print “Ja, Bedingung wurde erfüllt“
endif                    // für endif können Sie auch FI schreiben.
```

Sie können auch mehrere Bedingungen in die Abfrage integrieren, z.B.

```
If (x=2) then
        Print “Ja, Bedingung wurde erfüllt“
elseif(x=3)
        Print “Ja, Bedingung wurde erfüllt“
endif
```

Bei diesem Beispiel wird geprüft ob x=2 oder x=3. Je nachdem, welche Bedingung zu trifft, wird der entsprechende nachfolgende Code ausgeführt.

Wenn Sie jetzt einen Code in die Abfrage intrigieren wollen der immer ausgeführt wird wenn die beiden anderen Bedingungen nicht erfüllt werden, dann verwenden Sie einfach das Else Argument.

```
If (x=2) then
        Print “Ja, Bedingung wurde erfüllt“
elseif(x=3)
        Print “Ja, Bedingung wurde erfüllt“
else
        Print “Dieser Code wird immer ausgeführt, wenn eine der beiden
        Bedingungen nicht erfüllt werden.“
endif
```

Wenn Sie mehrere Bedingungen haben wird der Code der IF abfrage schnell unübersichtlich. Sie sollten dann auf die Switch Case Abfrage ausweichen.

10.2 Switch Case

Nun sind wir bei der Switch Case Abfrage. Diese ist bei mehreren Bedingungen übersichtlicher als die If elseif else Konstruktion.

Der Befehl SWITCH leitet die Abfrage ein und die in Klammern nachfolgende stehende Variable wird überprüft. Die eigentliche Überprüfung ob ein Fall eintritt oder nicht wird mit der CASE Anweisung gemacht. Man könnte auch sagen, das der Befehl SWITCH die zu überprüfende Variable zur Verfügung, für die CASE Anweisung, stellt.

Bei diesem Beispiel wird die Tastatur abgefragt.

```
switch(keymsg$)
  case "left"
    print "Sie haben die Taste links gedrückt"
  break
  case "ö"
    print "Sie haben die Taste ö gedrückt"
  break
  default:
  break
end switch
```

Der Break Befehl am Ende einer Case Anweisung dient dazu, das dann wirklich Schluß ist mit der Ausführung des Codes hinter der Case. Wenn Sie die Break Anweisung weglassen, würde das Programm alle Anweisungen auch alle anderen CASES) abarbeiten, solange bis das Programm auf einen Break oder auf End Switch trifft. Genauso wie bei der IF THEN Abfrage gibt es hier eine Möglichkeit, eine Anweisung zu definieren, wenn keine der Bedingungen erfüllt wird. Dazu benötigen Sie die Anweisung Default. Diese steht immer hinter den Cases, wie oben im Beispiel zu sehen ist. Sobald keine der Bedingungen erfüllt wurde und das Programm auf die Anweisung Default trifft, wird der darauf folgendeBefehl ausgeführt und in diesem Fall die Abfrage mit der Anweisung Break beendet. Eine Switch Abfrage wird immer mit End Switch beendet.

10.3 Weitere Operanten für Entscheidungen (and true false not continue)

In diesem Abschnitt werden Sie 4 weitere möglichen Operanten für die IF Schleife kennenlernen und eine weitere für While wend oder Do While Schleife.

Der Befehl **and** wird bei einer If Schleife dafür verwendet , das man zwei Bedingungen zusammenfügt, die dann erfüllt sein müssen.

```
If (x=10 and y=10) then
        print “yes“
else
        print “no“
endif
```

Mit Not können Sie eine If Abfrage verneinen.

```
If not (x=10 and y=10) then
        print “yes“
else
        print “no“
endif
```

Die If Abfrage prüft jetzt foglendes. Wenn x nicht 10 und y auch nicht 10 ist, dann mache etwas.

Mit False und True können Sie in einer IF Abfrage eine Variable auf einen x-beliebigen Wert prüfen, d.h Sie prüfen dann nur ob die Variable einen Wert hat oder nicht und **Nicht** auf den Wert der Variable.

Als Beispiel.

```
x=10

If (x=true) then
        print “yes“
else
        print “no“
endif
```

In dem obigen Beispiel wird yes ausgegeben, in dem nachfolgenden Beispiel wird no ausgegeben, da die Variable immer noch den Wert 10 hat und somit gefüllt ist. Gäbe es die Variable nicht oder wäre diese auf ““ gesetzt, so müßte das untere Beispiel yes ausgeben.

```
If (x=false) then
        print “yes“
else
        print “no“
endif
```

11. Funktionen definieren

Wofür werden Funktionen benötigt? Sie benötigen Funktionen immer dann, wenn sie bestimmte Programmabläufe immer wieder aufrufen, z.B. das Speichern einer Datei.
Das Speichern einer Datei erfordert, das Sie auf ein Button klicken oder eine Tastenkombination ausführen. In diesem Moment wird im Programm eine Funktion aufgerufen, die die Daten übermittelt bekommt oder wenn Sie global im Programm zur Verfügung stehen, sich selbst zieht. Dann wird der Code in der Funktion abgearbeitet, d.h. die Daten werden zur Speicherung in der Funktion aufgerufen und dann in die Datei geschrieben.

Als Beispielprogramm dient unsere Tabelle mit dem kleinen 1x1. Diese Tabelle wollen wir jetzt als CSV Datei speichern.

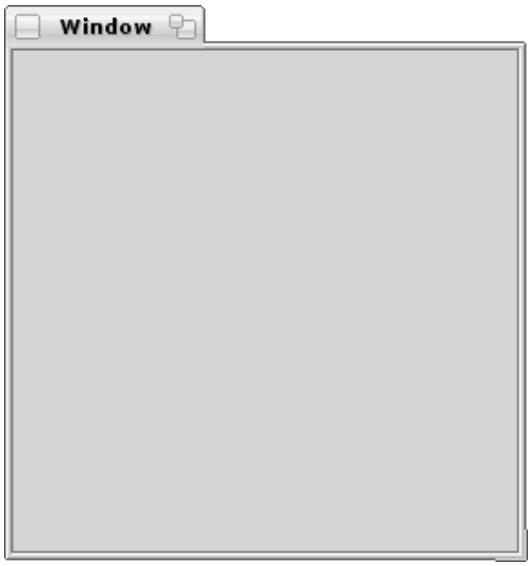
Programmcode:

Des Weiteren können Sie Funktionen auch in andere Dateien auslagern und als Libary benutzen. Diese Dateien können Sie mit Import einfügen. Export liefert ihnen die Daten an die Externe Funktion (Libary). Sie müssen in ihrer Libary daran denken, das wenn diese Libary ihnen eine Wert zurückliefern soll den Befehl
return(Wert_der_Zurückgegeben_werden_soll) einzubauen.

12. Die GUI Befehle

Gerade in der Anfangszeit, beim Programmieren lernen mit yab, stößt man oft an seine Grenzen. Dies liegt daran, dass man eine bestimmt Vorstellung hat, seine Ideen umzusetzen. Leider weiß man meistens nicht ob oder wie dieses umzusetzen ist. Diese Auflistung soll helfen, schneller eine benötigte Funktion zu finden, um dann in den yab Commands nach deren genauen Definition zu suchen. Bei allen Nachfolgenden Befehlserklärungen wird in Rot der Befehl dargestellt. In Blau die dazugehörigen Variablen bzw. Option und in Grün die zu den Optionen gehörenden Werte. Diese Werte (Value$) können auch mehrere Beinhalten. Wenn Sie mehrere Verwenden möchten, so müssen Sie diese dann mit einen Komma getrennt schreiben.

12.1 WINDOW



Das WINDOW, zu Deutsch FENSTER, ist der Grundbaustein einer jeden Benutzeroberfläche. Auf diesem werden alle Programmbausteine aufgesetzt.

WINDOW OPEN x1,y1 TO x2 ,y2, ID$, Title$

Öffnet ein Fenster an Position x1,y1 nach x2,y2 der Bildschirmkoordinaten mit dem Titel Title$. Der Platzhalter ID$ in der oberen Zeile steht für den Namen des Fenster, wie es im Programm angesprochen wird. Sie können für die Variable auch den Namen fest vergeben, diesen müssten Sie dann jedoch in Anführungszeichen schreiben, z.B.

WINDOW CLOSE ID$	Schließt ein Fenster, welches den Namen der Variablen WindowView$ enthält.

n = WINDOW COUNT	Gibt die Nummer des geöffneten Fensters zurück.

WINDOW SET Window$, Option$, Value$		Bei den Value angaben, kann man mehrere hinterander mit einem Komma getrennt schreiben. Nur sollte man darauf achten das diese sich gegenseitig aufheben.
Option$ = "Look"	Document Titled(default) Floating Modal Bordered No-Border	Siehe: "BeBook->Interface Kit->BWindow->Constants and Defined Types" für mehr Details.
Option$ = "Feel"	Normal(default) Modal-App Modal-All Floating-App Floating-All	Siehe: "BeBook->Interface Kit->BWindow->Constants and Defined Types" für mehr Details.
Option$ = "Flags"	Not-Closable Not-Zoomable Not-Minimizable Not-H-Resizable Not-V-Resizable Not-Resizable No-Workspace-Activation Accept-First-Click	Setzt das Fenster auf nicht schließbar Setzt das Fenster auf nicht Zoomable Setzt das Fenster auf nicht Minimizable (es kann nicht kleiner gemacht werden. Setzt die Fensterverstellmöglichkeiten so, dass man es in der Breite nicht verändern kann Setzt die Fensterverstellmöglichkeiten so, dass man es in der Höhe nicht verändern kann Setzt die Fensterverstellmöglichkeiten so, dass man es in der Höhe und der Breite nicht verändern kann
Option$ = "Flags"	"Reset"	
Option$ = "Workspace"	"All"	Setzt die flags zurück (auf none).
Option$ = "Workspace"	"Current"	Zeigt das Fenster auf allen Workspaces. Zeigt das Fenster in nur einer Workspace.

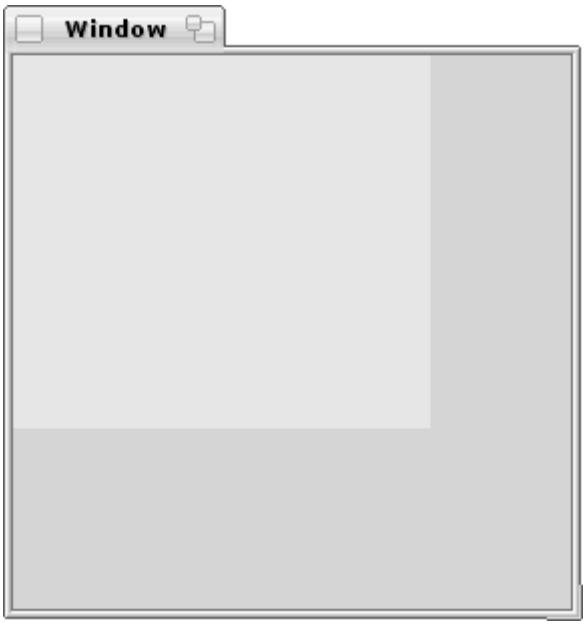
WINDOW SET Window$, Option$, r, g, b		
Option$ = "BGColor" Option$ = "HighColor" Option$ = "LowColor"	r,g,b (216,216,216 Vorgabe) r,g,b (0,0,0 Vorgabe) r,g,b (216,216,216 Vorgabe)	Setzt die Hintergrundfarbe auf die RGB Farben Setzt die Highcolor-Farbe auf die RGB Farben Setzt die Lowcolor-Farbe auf die RGB Farben

WINDOW SET Window$, Option$, x,y	
Option$ = "ResizeTo" Option$ = "MoveTo" Option$ = "MinimumTo" Option$ = "MaximumTo"	Setzt die

WINDOW SET WindowView$, Option$	
Option$ = "Activate"	Aktiviert das Fenster, sodaß es in den Vordergund kommt.
Option$ = "Deactivate"	Deaktiviert das Fenster, sodaß es in den Hintergrund rückt.
Option$ = "Minimize"	Minimiert das Fenster, oder stellt es wieder her, falls es vorher minimiert war
Option$ = "Maximize"	Maximiert (zoomt) das Fenster.
Option$ = "Enable-Updates"	Aktiviert die automatischen Fensterupdates.
Option$ = "Disable-Updates"	Schaltet die automatischen Fensterupdates ab

12.2 VIEW

Der VIEW ist eine auf ein WINDOW oder anderen VIEW gesetzte Ansicht. Diese Ansicht ist, wenn man diese nicht wie bei der Abbildung gezeigt, farblich abhebt, für den Anwender nicht sichtbar. Der Vorteil eines VIEWs ist, dass man die Möglichkeit hat einen Programmbereich (aufgesetzt auf den VIEW) zu entfernen, ohne dabei das Fenster schließen und wieder öffnen zu müssen.



VIEW x1,y1 TO x2 ,y2, ID$, View$

Fügt einen View hinzu.

VIEW DROPZONE View$	Definiert einen View als eine DropZone, welche dropped files akzeptiert. Die DROPZONE akzeptiert jetzt mehrere Dateien und gibt sie in umgekehrter Reihenfolge als message zurück.

Result = VIEW GET View$, Option$	
Option$ = "Position-X " Option$ = "Position-Y " Option$ = "Width" Option$ = "Height" Option$ = "Exists" Option$ = "Focused"	Gibt die angeforderte Information zurück. Falls es mit "Exists" oder "Focused" verwendet wurde ist 1 (zutreffend) und 0 (unzutreffend)

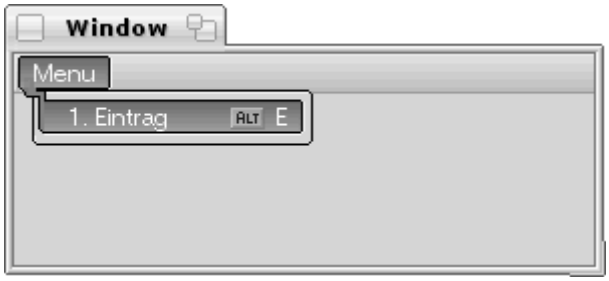
VIEW REMOVE View$	Entfernt den View

Achtung bei VIEW REMOVE.

Löscht alle internen Information über Menüs und Dropboxes.

Entfernen Sie niemals Menüs mit Verknüpfungen, andernfalls wird YAB abstürzen, wenn ein Tastenkürzel gedrückt wird!

12.3 MENU



Über den Befehl MENU kann man ein Auswahlmenü in das Programmfenster einbauen. Es sind MENUs und SUBMENUs möglich.

MENU Head$, Menu$, Shortcut$, View$

Fügt ein Menü in die Menubar ein mit dem Menüfeld Head$, dem Menuitem Menu$ und dem Kürzel Shortcut$ im View$.
Wenn Sie ein Tastaturkürzel zu dem Menüpunkt hinzufügen möchten, so können Sie die Variable Shortcut$ dafür benutzen.
Die Angabe bei dem Shortcut Shortcut$ kann am Anfang einen oder mehrere der Modifizierer haben, gefolgt vom dem Kürzelbuchstaben am Ende.

Modifiers$+ShortcutCharacter$

"S" -- Für die Shift Taste
"C" -- Für die Taste Steuerung
"O" -- Für die Optionen Taste (an den meisten Tastaturen wahrscheinlich die Windows Taste)

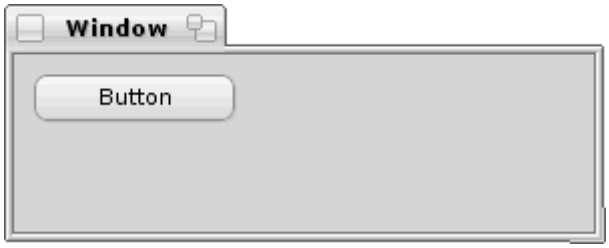
Diese Modifizierer können kombiniert werden, aber folgende Kombinationen funktionieren nicht: "SO", "SC" und "SCO"

Die Kommando Taste (*ALT*) ist immer Teil eines Tastenkürzels.

MENU SET MenuHead$, SetRadioMode, View$	Setzt das Menü auf radio mode, wenn ein Eintrag markiert wurde.
Sie müssen den ersten markierten Menüeintrag selber markieren	

MENU SET MenuHead$, MenuItem$, Option$, View$	
Option$ = "Disable" Option$ = "Enable" Option$ = "Mark" Option$ = "Plain" Option$ = "Remove"	Deaktviert einen Menüpunkt Aktviert einen Menüpunkt Markiert einen Menüpunkt Demarkiert einen Menüpunkt **Entfernt einen Menupünkt**

12.4 BUTTON



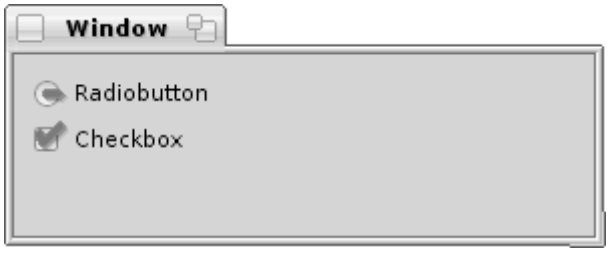
Ein BUTTON ist eine Schaltfläche, über welche man dem Anwender das Aktivieren eines Programmablaufes ausführen läßt. Neben dem klassischen BUTTON , ist es auch möglich eine Schaltfläche aus Grafiken darzustellen (BUTTON IMAGE).

BUTTON x1,y1 TO x2 ,y2, ID$, Label$, View$

Setzt einen button an Position (x1,y1) nach (x2,y2) mit Beschriftung Label$ im View View$

BUTTON IMAGE x,y , ID$, EnabledPressed$, EnabledNormal$, Disabled$, View$	Erstellt einen Grafischen Button (Image Button) bei (x,y) im View$ mit drei Dateien:
EnabledNormal$ EnabledPressed$ Disabled$	Enthält das Bild für einen nicht gedrückten Button Enthält das Bild für einen gedrückten Button Enthält das Bild für einen deaktivierten Button (Kann mit einem leeren String "" gesetzt werden, wenn kein deaktivierter Button benötigt wird.).

12.5 CHECKBOX & RADIOBUTTON



CHECKBOXen und RADIOBUTTONs finden in den meisten Fällen dann Verwendung, wenn man eine Programmabfrage oder Optionen vom Anwender ermitteln möchte.

CHECKBOX x1,y1 , ID$, Label$, IsActivated, View$

Anzeigen einer Checkbox an Position x1,y1 mit der Bezeichnung Label$ im View View$. Wenn IsActivated auf 0 gesetzt ist, dann ist die Checkbox aus, ansonsten an.

CHECKBOX SET CheckBox$, IsActivated	(De-)Aktivieren der Checkbox: CheckBox$.

CHECKBOX IMAGE x,y , ID$, EnabledOn$, EnabledOff$, DisabledOn$, DisabledOff$, IsActivated, View$	Erstellt eine Image Checkbox an Position x1,y1 im View: View$ mit vier Dateien
EnabledNormal$ EnabledPressed$ DisabledNormal$ DisabledPressed$ IsActivated	Enthält das Bild einer nicht gedrückten Checkbox Enthält das Bild einer gedrückten Checkbox Enthält das Bild einer deaktivierten Checkbox (Hier kann auch ein leerer String "" gesetzt werden) Enthält das Bild einer gedrückten, deaktivierten Checkbox (Hier kann auch ein leerer String "" gesetzt werden). Setze isActivated auf true|false wenn die Checkbox AN oder AUS sein soll.

RADIOBUTTON x1,y1 , ID$, Label$, IsActivated, View$

Fügt einen Radiobutton an Position x1,y1 mit der Bezeichnung Label$ im View View$ ein. Wenn isActivated auf 0 gesetzt ist, dann ist der Radiobutton aus, ansonsten ist er an. Radiobuttons sollten innerhalb eines Views gemeinsam gruppiert werden.

RADIOBUTTON SET RadioButton$, IsActivated
De- oder Activiert den Radiobutton RadioButton$.

12.6 TEXTCONTROL

Eine TEXTCONTROL ist eine Eingabefläche für den Anwender. Sie kann natürlich auch als Ausgabefläche verwendet werden, besitzt jedoch keine Scrollmöglichkeit.

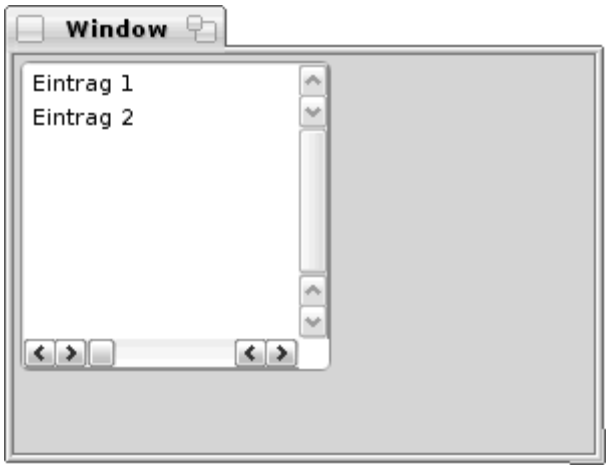
TEXTCONTROL x1,y1 TO x2 ,y2, ID$, Label$, Text$, View$

Öffnet ein Textcontrol von Position x1,y1 nach x2,y2 mit der Bezeichnung Label$ und stellt einen Text Text$ auf dem View View$ dar.

TEXTCONTROL SET TextControl$, Text$	Setzt den Textcontrol Text auf Text$.
TEXTCONTROL SET TextControl$, IsPassword	IsPassword = false -- Normale Eingabe IsPassword = true -- Verdeckte Eingabe
TEXTCONTROL CLEAR TextControl$	Löscht den Text eines Textcontrol.
Text$ = TEXTCONTROL GET$ TextControl$	Liest den Eintrag eines Textcontrol TextControl$ aus, auch wenn Enter nicht gedrÃückt wurde.

12.7 LISTBOX

Eine LISTBOX ist eine Liste, für Optionen oder Informationen, für den Andwender.



LISTBOX x1,y1 TO x2 ,y2, ID$, ScrollbarType, View$

Fügt eine leere Listbox an Position x1,y1 nach Position x2,y2 ein, bekannt als ID$ und mit einer ScrollbarType im View$.

ScrollbarType:
0 = keine
1 = vertikale Scrollbars
2 = horizontale Scrollbars
3 = vertikale & horizontale Scrollbars

LISTBOX CLEAR ListBox$	Löscht eine Listbox.
LISTBOX REMOVE ListBox$, Item$	Entfernt ein Item Item$ von der Listbox mit der Bezeichnung ID$.
LISTBOX SORT Listbox$	Sortiert die Listbox in alphabetischer Reihenfolge.
n = LISTBOX COUNT ListBox$	Zählt die Anzahl der Einträge der Listbox ListBox$.
Item$ = LISTBOX GET$ ListBox$, Position	Gibt ein Item von der Position Position zurück
LISTBOX SELECT ListBox$, Position	Wählt ein Item an Position Position. Mit Position = 0 wird diese als nicht markiert angesehen.
LISTBOX REMOVE ListBox$, Position	Entfernt ein Item an Position Position.
LISTBOX ADD ListBox$, Item$	Fügt das Item Item$ zur Listbox hinzu.
LISTBOX ADD ListBox$, Position, Item$	Fügt das Item Item$ an der Position Position hinzu.

12.8 DROPBOX

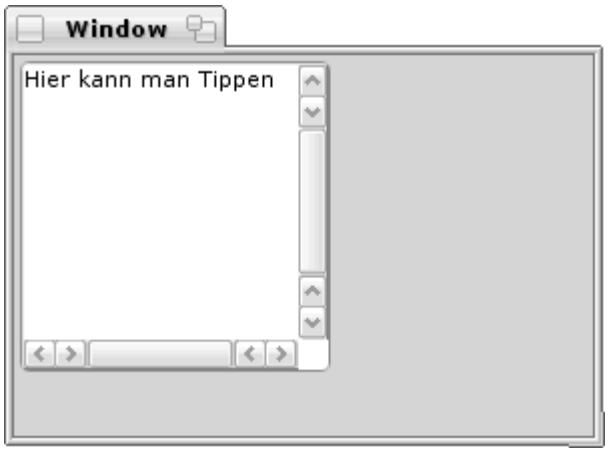
Eine DROPBOX ist eine Schaltfläche, die aufklappt und damit Platz für mehr als eine Option bietet wie beispielsweise der normale BUTTON.

12.9 TEXTEDIT

Die TEXTEDIT ist wie die TEXTCONTROL eine Eingabefläche für den Anwender. Der Unterschied zu TEXTCONTROL ist, dass die TEXTEDIT scrollbar ist.

TEXTEDIT x1,y1 TO x2 ,y2, ID$, ScrollbarType, View$

Öffnet einen Editor an Position x1,y1 nach x2,y2 mit ID$ (wird nicht angezeigt) im View$. Für die Scrollbartypes schauen Sie bitte bei ListboxEntry. Die TEXTEDIT folgt ebenfalls allen Seiten im Standard Layout.



Die Voreingestellten Werte und Optionen in einer Textedit sehen Sie in der folgenden Tabelle.

Vorgegebene Farben sind:

bgcolor = 255,255,255 (weiss)	Hintergrund der Textedit
textcolor = 0,0,0 (schwarz)	Textfarbe in der Textedit
color1 = 0,0,255 (blau)	Textfarbe für das erste Wort bei Auto-Vervollständigen
color2 = 255,0,0 (rot)	Textfarbe für das zweite Wort bei Auto-Vervollständigen
color3 = 0,250,0 (grün)	Textfarbe für das dritte Wort bei Auto-Vervollständigen
color4 = 185,185,185 (grau)	Textfarbe für das vierte Wort bei Auto-Vervollständigen
char-color = 200,0,255 (magenta)	Textfarbe für das erste Wort Zeichen eines Wortes

Vorgegebene Optionen in der Textedit sind:

autoindent = false	
wordwrap = true	Automatischer Zeilenumbruch
editable = true	Editierbar Aktiv
color-case-sensitive = false	
changed = false	
has-autocompletion = true	Auto-Vervollständigen Aktiv
autocomplete-start = 4	Ab den vierten Zeichen wird das Wort automatisch Vervollständigt

TEXTEDIT ADD TextEdit$, Text$	Fügt Text Text$ in den Textedit TextEdit$ im View View$ ein.
TEXTEDIT CLEAR TextEdit$	Löscht den Text des Textedit TextEdit$ im View View$.
EnteredText$ = TEXTEDIT GET$ TextEdit$	Eingegebener Text$ hält den Text von dem Textedit ID$ im View$
TextLine$ = TEXTEDIT GET$ TextEdit$, LineNumber	TextLine$ beinhaltet den Text aus der Zeile Linenumber
Width = TEXTEDIT GET$ TextEdit$, "Line-Width", LineNumber	Width beinhaltet die Breite des Textes aus der Zeile Linenumber
Height = TEXTEDIT GET$ TextEdit$, "Line-Height", LineNumber	Height beinhaltet die Höhe des Textes aus der Zeile Linenumber

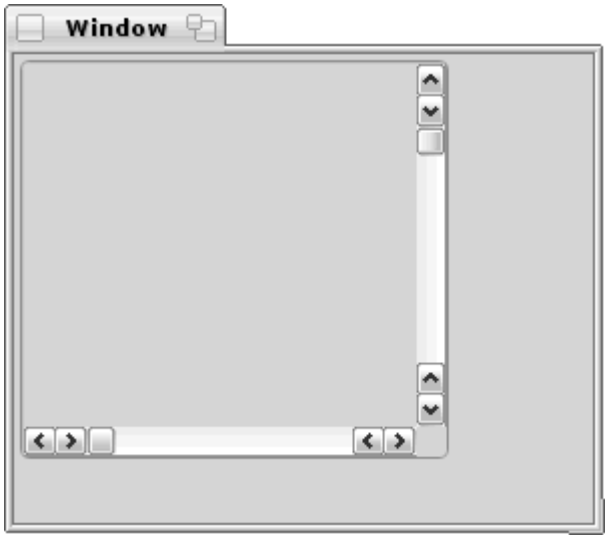
LineNumber = TEXTEDIT GET$ TextEdit$, Option$	
Option$ = "hasChanged"	Die Option$ "hasChanged" gibt die Zeile zurück in der sich etwas geändert hat
Option$ = "countlines"	Gibt die Anzahl der Zeilen in einer Textedit zurück
Option$ = "currentline"	Gibt die aktuelle Zeile in einer Textedit zurück
Option$ = "textlength"	Gibt die aktuelle Länge des Textes in einer Textedit zurück
Option$ = "cursor-position"	Gibt die aktuelle Position des Cursor in einer Textedit zurück
Option$ = "vertical-scrollbar"	Gibt den aktuellen Wert der Vertikalen Scrollbar in einer Textedit zurück
Option$ = "horizontal-scrollbar"	Gibt den aktuellen Wert der Horizontalen Scrollbar in einer Textedit zurück

TEXTEDIT SET TextEdit$, Option$, Value		Kann auch ohne Value angegeben werden, dann sind jeweils nur die letzten 4 Optionen möglich
Option$ = "autoindent"	Value = true oder false	
Option$ = "wordwrap"	Value = true oder false	Bricht die Zeile in der Textedit automatisch um
Option$ = "editable"	Value = true oder false	Setzt die Textedit auf Editierbar oder nicht
Option$ = "color-case-sensitive"	Value = true oder false	
Option$ = "changed"	Value = true oder false	
Option$ = "gotoline"	Value = LineNumber	Springt zu der angegebenen Zeile in der Textedit
Option$ = "select"	Value = LineNumber	Selektiert die angegebene Zeile in der Textedit
Option$ = "tabwidth"	Value = TabWidth	Setzt die Tabsprungweite in der Textedit
Option$ = "textwidth"	Value = TextWidth	Setzt die Textbreite in der Textedit bis diese automatisch in die nächste Zeile umbricht
Option$ = "autocomplete-start"		
Option$ = "has-autocompletion"		
Option$ = "autocomplete"		Fügt ein Wort Value$ zur Auto-Vervollständigen-Liste hinzu.
Option$ = "font"		Richtet Font Value$ ein (wie bei DRAW SET) - Standard ist "system-plain"

TEXTEDIT COLOR TextEdit$, Option$, Command$		
Option$ = "color1, color2, color3, color4, char-color"	Command$ = "DeinBefehl"	Fügt den Befehl Command$ zur Liste von Wörtern hinzu, die für SyntaxHighlighting geprüft werden. Das Char-Color verhält sich anders und Highlightet nur den ersten Buchstaben von Command$.

TEXTEDIT COLOR TextEdit$, Option$, r, g,b	Option$ = "bgcolor, textcolor, color1, color2, color3, color4, char-color" , r, g, b

12.10 SCROLLBARS



Mit Hilfe der SCROLLBARS kann man einen VIEW scrollbar machen.

SCROLLBAR ID$, ScrollbarType , View$

Macht View View$ scrollfähig.

SCROLLBAR SET Scrollbar$, "Vertical Position", Position

SCROLLBAR SET Scrollbar$, "Horizontal Position", Position

Setzt die Scrollbar an die Position Position - Vorgabe ist (0,0).

SCROLLBAR SET Scrollbar$, "Vertical Range", Min, Max
SCROLLBAR SET Scrollbar$, "Horizontal Range", Min, Max

Setzt die Scrollbars auf einen maximalen Bereich - Vorgabe ist (1000,1000).

Der Maximalwert wird der aktuellen Höhe / Breite des Views hizugefügt.

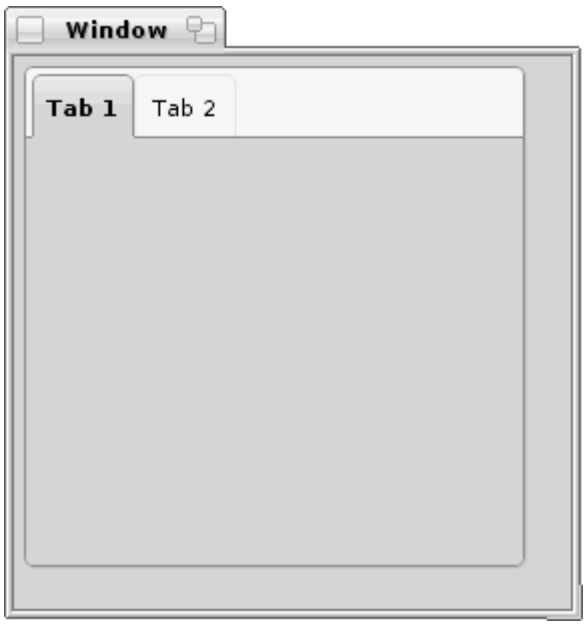
SCROLLBAR SET Scrollbar$, "Vertical Steps", SmallSteps, BigSteps
SCROLLBAR SET Scrollbar$, "Horizontal Steps", SmallSteps, BigSteps

Setzt die Scrollbar Schrittweite. SmallSteps sind die scrolling Steps wenn der Benutzer auf die Pfeilbuttons klickt - (Vorgabe ist 1,0). BigSteps sind die scrolling Steps wenn der Benutzer auf einen freien Bereich der scrollbar klickt - (Vorgabe ist 10.0).

Position = SCROLLBAR GET Scrollbar$, "Horizontal"
Position = SCROLLBAR GET Scrollbar$, "Vertical"

Ermitteln der aktuellen Position der Scrollbars.

12.11 TABVIEW



Ein TABVIEW ist ein VIEW, welcher mit einem TAB (Reiter) versehen ist. Dies hat den Vorteil, daß man mehrere VIEWs übereinander legen kann. Über das Anklicken eines der TABs, kann man dann in den angewählten TABVIEW wechseln. Durch die Möglichkeit die TABs zu Beschriften, wird das ganze dann auch noch sehr übersichtlich.

TABVIEW x1,y1 TO x2 ,y2, ID$, Option$, View$

Das Option$ = "top, bottom" ist ZETA only, Benutzer der R5 müssen dieses zwar auch setzen, aber die R5 Tabs sind immer oben, das ist vorgegeben.

TABVIEW ADD TabView$, TabName$

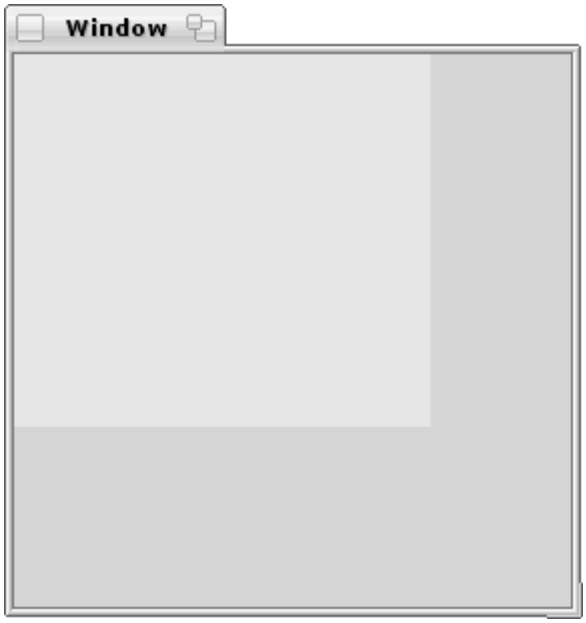
Für jeden Tab wird ein neuer View erstellt. Sie können Widgets hinzufügen oder Draw benutzen wie gehabt. Die IDs für die Views ist TabView$+"1" für den ersten, TabView$+"2" für den zweiten usw.

TABVIEW SET TabView$, TabNumber

Öffnet den Tab mit Nummer TabNumber.

TabNumber = TABVIEW GET TABVIEW$
Auslesen des aktuell geöffneten Tabs.

12.12 STACKVIEW



Ein STACKVIEW ist ein Stapel von VIEWs, die übereinander liegen und dann untereinander mit Befehlen gewechselt werden können.

STACKVIEW x1,y1 TO x2 ,y2, ID$, NumberOfViews, View$

Setzt einen Stack auf einen Views auf, wobei immer nur einer sichtbar ist.

STACKVIEW SET StackView$, ViewNumber

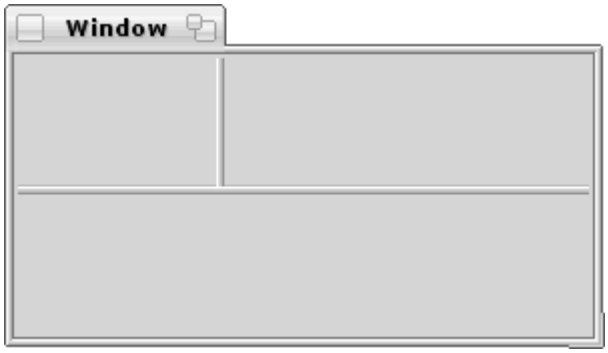
Setzt die Nummer des View ViewNumber als den sichtbaren View.

ViewNumber = STACKVIEW GET StackView$

Auslesen der Nummer des aktuell sichtbaren Views.

12.13 SPLITVIEW

Ein SPLITVIEW ist ein VIEW, der in mehrere Teile gesplittet wird. Der View kann Waagerecht, Senkrecht oder Waagerecht und Senkrecht gesplittet werden. Hierbei kann man zwischen festen oder variablen Größen wählen.



SPLITVIEW x1, y1 TO x2 ,y2, ID$, IsVerticalSplit, NormalStyle, View$

Setzt einen neuen View mit der Bezeichnung ID$ und teilt ihn in zwei neue Views ID$+"1" und ID$+"2".

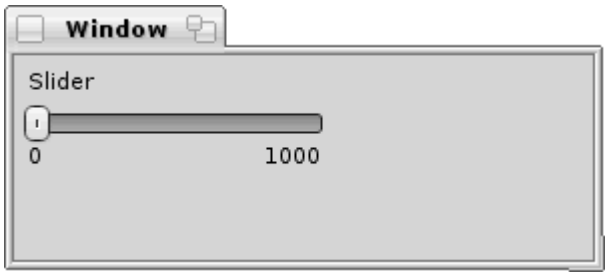
SPLITVIEW x1,y1 TO x2 ,y2, ID$, IsVerticalSplit, NormalStyle, , View$	Setzt einen neuen View mit der Bezeichnung ID$ und teilt ihn in zwei neue Views ID$+"1" und ID$+"2"
IsVerticalSplit = true IsVerticalSplit = false NormalStyle = true NormalStyle = false	wird ein vertical splitter gesetzt wird ein horizontal splitter gesetzt dann ist der splitter 10 Pixel stark. dann ist der splitter 4 Pixel stark.

SPLITVIEW SET SplitView$, "Divider", Position	Setzt die Position des Divider, Voreinstellung ist die Hälfte des gesamten Splitview.

SPLITVIEW SET SplitView$, "MinimumSizes", LeftOrTop, RightOrBottom	Setzt die minimum Größe (Minimumsize) vom linken (oder oberen) View und dem rechten (oder unteren) View. Wichtig ist dabei, das Sie sich im klaren sind, welchen Splitview Sie gerade einstellen.

Position = SPLITVIEW GET SplitView$, "Divider"	Auslesen der aktuellen Position des Divider.

12.14 SLIDER



SLIDER werden verwendet wenn man dem Anwender eine Angabe einer Ausgabe ermöglichen möchte.

SLIDER x1,y1 TO x2 ,y2, ID$, Label$, Min, , Max, View$

Fügt einen Slider mit einem Minimum- und einem Maximumwert hinzu (Min, Max).

SLIDER x1,y1 TO x2 ,y2, ID$, Label$, Min , Max, Option$, View$	
Option$ = "block" Option$ = “triangle” Option$ = “horizontal” Option$ = “vertical"	

SLIDER LABEL Slider$, StartLabel$, EndLabel$	Setzt Start- und Endbegrenzung der Beschriftung.

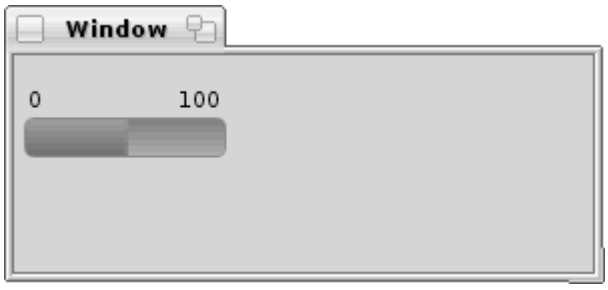
SLIDER SET Silder$, Location$, Count	
Location$ = "none” Location$ = "Bottom” Location$ = "Top” Location$ = "Both” Location$ = "Left” Location$ = "Rigth” Count	nutzt Hashmarken für horizontale und vertikale Slider. Hashmarken sind Schrittmarken in dem Slider. Nummer der Hashmarken

SLIDER COLOR Silder$, Part$, r,g ,b	Hiermit können Sie die Silderfarben anpassen
Part$ = "barcolor" Part$ = "fillcolor"	

SLIDER SET Silder$, Value	Setzt den Wert des Slider

Value = SLIDER GET Slider$	Holt den aktuell gewählten Wert des Slider

12.15 STATUSBAR



Die STATUSBAR findet dann Verwendung, wenn man den Status eines Ablaufes darstellen möchte.

STATUSBAR x1,y1 to x2 ,y2,ID$, Label1$,Label2$, View$

Erstellt eine Statusbar mit der Bezeichnung ID$ und den beiden Wertebezeichnungen Label1$ und Label2$ auf den View$.

Label1$ ist die Bezeichnung, die links an der Statusbar angezeigt wird und Label2$ ist die Bezeichnung, die rechts an der Statusbar angezeigt wird.

STATUSBAR SET ID$,Label1$,Label2$,State

Setzt die Statusbar mit der Bezeichnung ID$ auf einen Wert State und ändert die Bezeichnungen Label1$ und Label2$.

12.16 TREEBOX

Eine TREEBOX ist ein sogenanntes Baummenü. Hier bekommt man die Ordnerstuktur, welche man angibt oder ermitteln läßt so ausgegeben, daß diese aus dem Menü heraus ersichtlich ist. Hauptordner werden ganz links und Unterordner nach rechts versetzt zum Hauptordner dargestellt.

TREEBOX x1,y1 to x2 ,y2, ID$, ScrollbarType, View$

Fügt eine Treebox hinzu. Ist im großen und ganezen genauso wie eine Listbox, nur das sie einen Dateibaum anzeiogen können.

ITEM ADD funktioniert hier nicht. Benutzen Sie stattdessen TREEBOX ADD. ITEM ADD wird in den nächsten Releases eingefügt.

TREEBOX ADD TreeBox$, RootItem$	Fügt das Item RootItem$ an die oberste Stelle des Trees.
TREEBOX ADD TreeBox$, HeadItem$, ListItem$, IsExpanded	Fügt das Item ListItem$ unter dem Level HeadItem$ des Trees hinzu, mit der Option ausgeklappte Treebox ja / nein
TREEBOX CLEAR TreeBox$	Löscht den Tree.
TREEBOX REMOVE TreeBox$, ListItem$	Entfernt den ersten Listeneintrag ListItem$ vom Tree. *Enthaltene Subitems werden auch entfernt.*
n = TREEBOX COUNT TreeBox$	Gibt die Anzahl der Einträge der TreeBox$ zurück.
Item$ = TREEBOX GET$ TreeBox$, Position	Ermittelt Item$ mit TREEBOX GET$ von TreeBox$.
TREEBOX EXPAND TreeBox$, Head$	Expandiert den Konten Head$ in der TreeBox$.
TREEBOX COLLAPSE TreeBox$, Head$	Schließt den Konten Head$ in der TreeBox$.
TREEBOX REMOVE TreeBox$, Position	Entfernt den Eintrag an der Position Position in TreeBox$.
TREEBOX REMOVE TreeBox$, Head$, ListItem$	Entfernt ListItem$ unter Head$ in TreeBox$.
TREEBOX SELECT TreeBox$, Position	Markiert den Eintrag an Position Position in TreeBox$.
TREEBOX SORT Treebox$	Sortiert die TreeBox$ alphabetisch

12.17 TEXTURL

Eine TEXTURL (Link) ist ein Text, durch dessen Anklicken eine Funktion ausführt wird.

TEXTURL x,y , Texturl_ID$, Label$, Address$, View$

Setzt die Web|Email|Ftp Addresse an Position x,y mit dem Label Label$ und der URL Address$.

Wenn Sie eine Email Adresse angeben so kann diese wie folgt geschrieben werden:

"mailto:foo@bar.com" oder foo@bar.com

Eine Web URL beginnt entweder mit "http://" oder mit "file://" und eine FTP Addresse beginnt mit "ftp://".

TEXTURL COLOR TextURL$, Option$, r, g,b	Setzt die Farbe für die URL.
"Label" "Click" "Mouse-over"	Setzt die Farbe des Labels. Setzt die Farbe des Labels, wenn es gedrückt wurde. Setzt die Farbe des Labels, wenn die Maus über das Label bewegt wird.

12.18 BOXVIEW

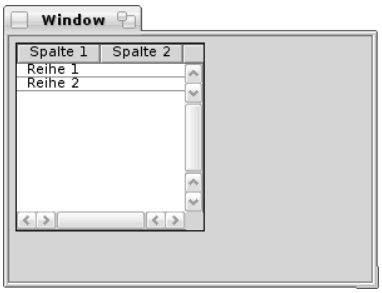
Ein BOXVIEW ist ein View, welcher durch eine Umrahmung und einer sichtbaren Beschriftung hervorgehoben wird.

BOXVIEW x1,y1 TO x2 ,y2, ID$, Text$, LineType, View$

Fügt ein neuen BoxView hinzu.

BOXVIEW x1,y1 TO x2 ,y2, ID$, Headline$, LineType, View$	Der aktuelle View eines BOXVIEW (wo Sie Ihre Widgets einbringen) ist bei (x1+5,y1+10) nach (x2-5,y2-5).
ID$ Headline$ LineType =0 LineType =1 LineType =2	Interner Bezeichner Angezeigter Name der Boxview keinen Rahmen einfache Linie heißt Rahmen, fancy line

12.19 COLUMNBOX



Eine COLUMNBOX ist eine frei definierbare Tabelle.

COLUMNBOX x1,y1 TO x2 ,y2, ID$, HasHScrollbar, Option$, View$

Erstellt eine Columnbox an Position x1,y1 to x2,y2 mit der ID ID$ auf einen View oder Window.

COLUMNBOX x1,y1 TO x2 ,y2, ColumnBox_ID$, HasHScrollbar, Option$, View$	Die Optionen können kombiniert werden, nur sollte man bedenken, das sich die Optionen nicht aufheben
HasHScrollbar = true oder 1 Option$ = “movable“ Option$ = “resizable“ Option$ = “removable“ Option$ = "popup"	Setzten sie den Wert, wenn Sie eine horizontale Scrollbar benötigen Die Spalten können in ihrer Anordnung vom User per Drag n Drop verändert werden. Die Spalten können in ihrer Breite vom User verändert werden. Der User kann die Spalten entfernen
COLUMNBOX COLUMN ColumnBox_ID$, Name$, Position, MaxWidth, MinWidth, Width, Option$	Fügt eine Spalte mit Namen Name$ der Columnbox Columnbox_ID$ hinzu
Position MaxWidth MinWidth Width Option$ = "align-left” Option$ = ”align-center” Option$ = “align-right"	Position der Spalte in der Columnbox Maximale Breite der Spalte auf die der User vergrößern kann Minimale Breite der Spalte auf die der User vergrößern kann Breite der Spalte beim ersten anzeigen Position des angezeigten Inhalts Linksbündig Mittig Rechtsbündig

COLUMNBOX ADD ColumnBox$, Column, Row, Height, Item$	Wenn Sie bei Item$ eines der folgenden Möglichkeiten angeben, wird auch ein Bild angezeigt
Item$ = "__Icon__="+FileName$ Item$ = "__Path__="+FileName$ Item$ = "__Mime__="+Signature$	wird das Bild FileName$ angezeigt. wird das große Trackericon angezeigt wird das kleine Icon des Mimetypes Signature$ angezeigt.

COLUMNBOX COLOR ColumnBox$, Option$, r, g,b	Die Optionen können kombiniert werden, nur sollte man bedenken, das sich die Optionen nicht aufheben
Option$ = "selection-text” Option$ = "non-focus-selection” Option$ = "selection” Option$ = ”text” Option$ = "row-divider” Option$ = "background" r, g,b	

COLUMNBOX SELECT ColumnBox$, Row	Wählt die angegebene Reihe ROW der Comumnbox aus.

COLUMNBOX REMOVE ColumnBox$, Row	Entfernt alle Einträge aus angegebene Reihe Row der ColumnBox$. Row = 0 bedeutet das diese deaktiviert ist.

COLUMNBOX CLEAR ColumnBox$	Entfert alle Einträge aus der ColumnBox$.

n = COLUMNBOX COUNT ColumnBox$	Ermitteln der Einträge (n) in der ColumnBox$.

Item$ = COLUMNBOX GET$ ColumnBox$, Column, Row	Item$ = Beinhaltet den ausgewählten Wert aus der Reihe Row und der Spalte Column

12.20 FILEPANEL

Ein FILEPANEL ermöglicht ein, durch den Anwender durchgeführtes, festsetzen eines Zieles im System. Dies wird zum Beispiel dann verwendet, wenn man dem Anwender die Möglichkeit geben möchte, den Speicherort einer Datei selber festzulegen. Gleiches gilt für das Laden einer Quelldatei.

File$ = FILEPANEL Mode$, Title$, Directory$

File$ = FILEPANEL Mode$, Title$, Directory$	File$ beinhaltet die von ihnen gewählte Datei
Mode$ = "Load-File" Mode$ = "Load-Directory" Mode$ = "Load-File-and-Directory" Mode$ = "Save-File" Title$ = “Ihr Name” Directory$ = “Ihr Verzeichnis”	Das Filepanel wird im *Laden Modus* geöffnet Das Filepanel wird im *Modus Verzeichnisauswahl* geöffnet Das Filepanel wird im *Modus Selektiere Verzeichnis und Datei und lade die Datei* geöffnet Das Filepanel wird im *Speichern Modus* geöffnet Beinhaltet den Namen des Filepanels. Beinhaltet ihr Verzeichnis in der die Datei gespeichert oder von der geladen werden soll

File$ = FILEPANEL Mode$, Title$, Directory$, Filename$	Gleiche Funktionsweise wie oben, nur das hier gezielt eine Datei oder Verzeichnis vorgegeben wird.

12.21 SPINCONTROL

Eine SPINCONTROL ist die Möglichkeit, dem Anwender eine Auswahl einer Anzahl auswählen zu lassen.

SPINCONTROL x,y , ID$, Label$, Min, Max , Step, View$

Erstellt eine SpinControl an Position x,y mit der ID ID$ auf einen View oder Window.

SPINCONTROL x,y , ID$, Label$, Min, Max , Step, View$	
Label$	Definiert die Bezeichnung der Spincontrol für den User (im obigen Beispiel heißt die Spincontrol *Spincontrol*)
Min	Definiert den niedrigsten Wert
Max	Definiert den höchsten Wert
Step	Setzt die Schrittweite fest, mit der der User die angezeigte Zahl über die Pfeiltasten verändern kann.
View$	Ist der View oder Window auf dem die Spincontrol liegt

SPINCONTROL SET SpinControl$, Value	Setzt den Wert Value der SpinControl$

Value = SPINCONTROL GET SpinControl$	Auslesen des aktuellen SpinControl Wertes.

12.22 CALENDAR

Mit CALENDAR hat man die Möglichkeit einen Kalender darzustellen. Hierfür gibt man lediglich die Position auf dem VIEW an und definiert das aktuelle oder gewünschte Datum.

CALENDAR x,y , ID$, Format$, Date$, View$

Öffnet ein Kalender Widget an Position (x,y) mit dem internen Bezeichner ID$. Setzt das Format und Datum und gibt das im View View$ aus.

CALENDAR x,y, Calendar_ID$, Format$, Date$, View$	
Format$ = ("DDMMYYYY" oder " MMDDYYYY") + ("." oder "/" oder "-")	Hiermit stellen Sie das Format inkl. Trennzeichen ein. Voreinstellung ist: "DDMMYYYY."
Date$	Hier geben sie das Startdatum in dem zuvor definierten Format vor

CALENDAR SET Calendar_ID$, Date$	Setzt das Datum mit dem gesetzten Format (DDMMYYYY oder MMDDYYYY).

Date$ = CALENDAR GET$ Calendar_ID$	Ermittelt das ausgewählte Datum aus Calendar_ID$ und speichert dieses in Date$.

12.23 CANVAS

Eine CANVAS ist ein VIEW, der durch seine besonderen Eigenschaften besonders für die Programmierung von Spielen geeignet ist, denn auf einer CANVAS kann flackerfrei gezeichnet werden.

CANVAS x1,y1 to x2 ,y2, ID$, View$

Eine Canvas ist ein spezieller View der für flackerfreies darstellen verwendet wird. Außerdem zeigt er ein nicht in der Größe veränderbaren Bitmap. Da hier keine Darstellung nacheinander gezeichneter Drawbefehle benötigt werden, wird auch kein DRAW FLUSH benötigt. Die Canvas hat immer einen weißen Hintergrund.

12.24 MESSAGE

msg$ = MESSAGE$

Sammelt die Nachrichten, welche durch GUI Elemente generiert werden.

Arrived = MESSAGE SEND Application$, Message$

Sendet eine Zeichenkette zur yab Anwendung mit der Signatur Application$ (vorgegebene Signatur ist: "application/x-vnd.yab-app", Sie können die Signatur durch Einfügen eines Kommentars in der ersten oder zweiten Zeile Ihres Programms ändern. Beispielsweise: // mimetype "application/x-vnd.myapp"

Funktioniert zur Zeit nicht für gebundene (bound) Programme!

Die Ziel-YAB-Anwendung wird eine Nachricht message$ erstellen, die sagt:" _Scripting:...|" wobei die ... die Message$ sind.

Das Kommando gibt folgende Fehlercodes zurück:
0: -- Nachricht wurde geliefert
1: -- Zielprogramm wurde nicht gefunden
2: -- Nachrichtenwarteschlange das Zieles ist voll
3: -- Time out, die Nachricht kam beim Ziel nicht an
4: -- Ein anderer Fehler trat auf

12.25 Maus MESSAGE

Mouse$ = MOUSE MESSAGE$(View$)

Gibt den Status der Maus bezüglich des View$ zurück. Es besteht aus X:Y:LMB:MMB:RMB (wobei MB die entsprechende linke, mittlere, rechte Maustaste ist).

Mouse$ = MOUSE MOVE$

Gibt Informationen über die Bewegung der Maus Mouse$ zurück (funktioniert auf *Views,Buttons, PictureButtons, ColumnViews*). Mouse$ hat dann einen der folgenden Werte.

ControlName:_InsideView
ControlName:_EnteredView
ControlName:_ExitedView
ControlName:_MouseDown
ControlName:_MouseUp
ControlName:_LeftMouseButton
ControlName:_RightMouseButton
ControlName:_MiddleMouseButton

ControlName und *ID* der Aktion

MOUSE SET Option$

Zeigt oder Verbirgt den Mousezeiger.

Option$ = "Hide" -- verbirgt den Mousezeiger
Option$ = "Show" -- zeigt den Mousezeiger
Option$ = "Obscure" -- verbirgt den Mousezeiger, wenn dieser bewegt wird.

13. Weitere grafische Befehle

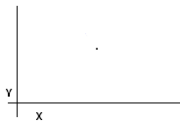
Die meisten grafischen Befehle haben hier immer Koordinatenangaben, dabei bezieht sich X auf die Waagerechte und Y auf die Senkrechte Position.

13.1 Draw Flush

Löscht alle Draw Elemente auf dem View

13.2 Draw Dot

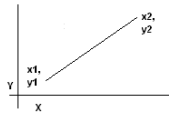
DRAW DOT x1,y1, View$



Zeichnet einen Punkt an der Koordinate x,y

13.3 Draw Line

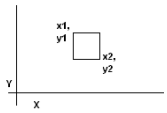
DRAW LINE x1,y1 TO x2,y2, View$



Zeichnet eine Linie von der Koordinate x1,y1 bis zu der Koordinate x2,y2.

13.4 Draw Rect

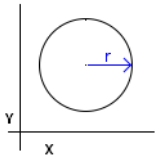
DRAW RECT x1,y1 TO x2,y2, View$



Zeichnet ein Rechteck von der Koordinate x1,y1 bis zu der Koordinate x2,y2.

13.5 Draw Circle

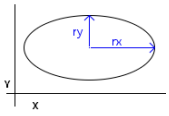
DRAW CIRCLE x,y,r, View$



Zeichnet einen Kreis am Mittelpunkt x,y mit einen Radius von r. Dieser Kreis ist erstmal nicht gefüllt.

13.6 Draw Ellipse

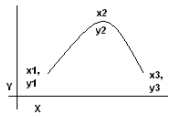
DRAW ELLIPSE x1,y1, xRadius, yRadius, View$



Zeichnet eine Ellipse am Mittelpunkt x,y mit einen xRadius und einen yRadius. Diese Ellipse ist erstmal nicht gefüllt.

13.7 Draw Curve

DRAW CURVE x1,y1, x2,y2, x3,y3, x4,y4, View$



Zeichnet eine Kuve zu den angegeben Koordinaten

13.8 Draw Polygone

N/A

13.9 Draw Bitmap

DRAW BITMAP x,y, Bitmap$, Mode$, View$

Zeichnet eine BITMAP auf einen View, oder eine BITMAP auf einer anderen BITMAP
Die möglichen Mode$ - Optionen sind:
"Copy" – Zeichnet die BITMAP ohne Transparenz auf dem entsprechenden View
"Alpha" -- Zeichnet die BITMAP mit Transparenz auf dem entsprechenden View

DRAW BITMAP x1,y1 TO x2,y2, Bitmap$, Mode$, View$

Zeichnet oder skaliert eine BITMAP in der Größe von x1, y1 zu x2,y2.

Wenn Sie die BITMAP skaliert darstellen wollen, so haben Sie die nachfolgenden Möglichkeiten:
Wenn Sie x2 auf -1 setzen, wird die Breite entsprechend der Höhe angepasst.
Wenn Sie y2 auf -1 setzen, wird die Höhe entsprechend der Breite angepasst.
Wenn Sie x2 und y2 auf -1 setzen, wird die BITMAP nicht skaliert. Entsprechend könnten Sie dann auf den Befehl DRAW BITMAP x,y, Bitmap$, Mode$, View$ verwenden.
Die möglichen Mode$ - Optionen sind:
"Copy" – Zeichnet die BITMAP ohne Transparenz auf dem entsprechenden View
"Alpha" -- Zeichnet die BITMAP mit Transparenz auf dem entsprechenden View

13.10 Draw Image

LoadError = DRAW IMAGE x,y, ImageFile$, View$

Zeichnet das Bild an der position x,y auf dem angegebenen View. Der Befehl gibt Statusmeldungen zu dem Ladezustand zurück, diese wären wie folgt:
LoadError:
0 = success (Erfolgreich)
1 = file not found (Datei nicht gefunden)
2 = translator roster not found (Die Umwandlungsbiliothek wurde nicht gefunden)
3 = translation failed (Die Umwandlung ist fehlgeschlagen)
4 = detaching bitmap failed (Mit der Auflösung des Bildes gibt es Probleme)

err = DRAW IMAGE x1,y1 TO x2,y2, Image$, View$

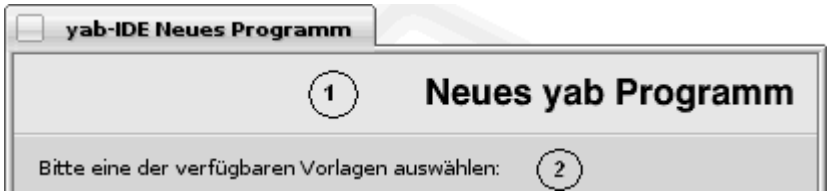
Zeichnet oder skaliert das Bild Image$ in der Größe von x1, y1 zu x2,y2. Transparente Bilder werden auch mit der Transparenz dargestellt.

Wenn Sie das Bild skaliert darstellen wollen, so haben Sie die nachfolgenden Möglichkeiten:
Wenn Sie x2 auf -1 setzen, wird die Breite entsprechend der Höhe angepasst.
Wenn Sie y2 auf -1 setzen, wird die Höhe entsprechend der Breite angepasst.
Wenn Sie x2 und y2 auf -1 setzen, wird das BILD nicht skaliert. Entsprechend könnten Sie dann auf den Befehl DRAW IMAGE x1,y1, Image$, View$ verwenden.

13.11 Draw Text

DRAW TEXT x1,y1, Text$, View$

Zeichnet einen Text an den angegeben Koordinaten. Siehe im nachfolgenden Bild Position 1 und 2.



13.12 Draw Set

Ermöglicht Ihnen diverse Einstellung vorzunehmen. Sie können z.B. die Hintergrundfarbe, Vordergrundfarbe, Textfarbe usw. einstellen.

DRAW SET FillOrStroke, Pattern$

Mit DRAW SET FillOrStroke können Sie Rechtecke, Ellipsen, Kreise und Polygone gefüllt darstellen.
FillOrStroke = 0 gefüllt (Fill)
FillOrStroke = 1 mit einem Pinsel gefüllt (stroke)
Pattern$ ist ein 8x8x1 großes Bit Map
Pattern$ = "HighSolidFill":Das zu füllende Objekt wird mit der eingestellten HighColor gefüllt
Pattern$ = "LowSolidFill": Das zu füllende Objekt wird mit der eingestellten LowColor gefüllt
Pattern$ = "CheckeredFill": gemusterte Füllung

DRAW SET Color$, r,g,b, View$

Mit DRAW SET Color$ können Sie die Hintergrundfarbe, die Hohe- und die niedrige Farbe für einen View einstellen. Die Grundwerte werden die nachfolgenden Dargestellt:
"BGColor", r,g,b (216,216,216 default)
"HighColor", r,g,b (0,0,0 default)
"LowColor", r,g,b (216,216,216 default)

DRAW SET Color$, Option$, View$

Mit DRAW SET Color$, Option$ können Sie für das Menü, den Panel bei Fileopen (Filesave) und die Ränder farblich einstellen.Welche Farbe ,ob nun High, Low oder BG-Color sie einstellen möchten, definieren Sie mit der Variablen Color$. Die möglichen Einstellungen für die Variable Option$ sehen Sie in der nachfolgenden Auflistung:

"Panel-Background-Color"
"Panel-Text-Color"
"Panel-Link-Color"
"Menu-Background-Color"
"Menu-Selection-Background-Color"
"Menu-Item-Text-Color"
"Menu-Selected-Item-Text-Color"
"Keyboard-Navigation-Color"
"Jan-Favorite-Color"
To those options the following can be added:
"Lighten-1-Tint"
"Lighten-2-Tint"
"Lighten-Max-Tint"
"Darken-1-Tint"
"Darken-2-Tint"

"Darken-3-Tint"
"Darken-4-Tint"
"Darken-Max-Tint"
"Disabled-Label-Tint"
"Disabled-Mark-Tint"
"Highlight-Background-Tint"

Beispiel: draw set "bgcolor", "Menu-Background-Color, Lighten-1-Tint", View$

DRAW SET Font$, View$

Mit diesem Befehl können Sie die Schriftart, den Schrfitstill und die Größe einstellen. Die Einstellungen bezieht sich dann immer auf den angegeben View.

Font$ = FontFamily + "," + FontStyle + "," + Size
Font$ = "system-plain" für den Systemweiten eingestellten Font
Font$ = "system-bold" für den Systemweiten FETT eingestellten Font
Font$ = "system-fixed" für den Systemweiten fixierten eingestellten Font

DRAW SET "Alpha", Alpha-Value

Mit diesem Befehl setzten Sie den Alpha Channel für die Transparenz der gezeichneten Farben. Wenn Sie eine Transparenz benötigen, nutzen Sie bitte diesen Befehl bevor Sie mit Draw Color ihre Farbeinstellungen vornehmen.

Alpha-Wert kann ein Wert zwischen 0 und 255 sein, wobei 0 für die totale Transparenz und 255 für Undurchlässig steht.

Hinweis: Wenn der Alpha Wert kleiner als 255 ist, sollten Sie die Funktion HighSolidFill nutzen, Objekte die mit gemustert (Pattern) dargestellt werden, werden ignoriert.
Hinweis: Transparente Zeichnungen, wo der Alpha-Wert kleiner als 255 ist, werden nicht korrekt gedruckt.

13.13 Draw Get

Draw GET ermöglicht Ihnen diverse Informationen von anderen Draw Befehlen abzurufen. Sie können z.B. herausfinden welchen Sie verwenden, bzw. voreingestellt im System ist und die entsprechende Höhe und Breite eines Textes.

FontFamilies$ = DRAW GET$ "FontFamily"

Gibt die Font Familie zurück, d.h. Sie wissen dann, wie der Font heißt der eingestellt worden ist.

FontStyles$ = DRAW GET$ FontFamily$

Gibt die Font Stil zurück, d.h. Sie wissen dann, wie der Font dargestellt wird. Derzeit mögliche Ergebnisse sollten Italic (Schrägschrift), Bold (Fett) und Underline (Unterschrichen) sein.

Height = DRAW GET "Max-Text-Height", View$

Gibt die Höhe des kompletten Textes auf dem View View$ zurück.

Width = DRAW GET "Text-Width", Text$, View$

Gibt die Breite des kompletten Textes auf dem View View$ zurück.

14. Dateioperationen

End of File (Datei-Ende)
Wie Sie sehen, durchläuft das Programm eine Schleife, solange bis die Datei vollständig gelesen wurde. Dies wurde erreicht mittels der Datei-Ende-Funktion eof(1), die FALSCH zurückgibt, solange noch Zeichen in der Datei sind, deren Dateinummer als Argument genannt wurde, und gibt WAHR zurück, wenn das Ende der Datei erreicht wird. Als einen besonderen Fall können Sie eof(0) benutzen um zu prüfen, ob noch Zeichen beim Standard Input (z.B. Tastatur) vorhanden sind. Dies macht möglicherweise nur Sinn, wenn Yabasic als Script läuft.

Open
Close

Tell
Seek

Wahlfreier Dateizugriff
Normalerweise liest Yabasic ein Byte nach dem anderen aus einer Datei. Dennoch können Sie die Funktionen seek() und tell() benutzen, um sich in einer geöffneten Datei frei zu bewegen:

Diese kleine Beispielprogram soll ihnen zeigen, wie man sich frei in einer Datei bewegt.

```
REM Zuerst erstellen wir eine Datei:

open #1,"test.dat","w"
        print #1 "abcdefghijkl"
close #1

REM Dann oeffnen wir die Datei:

open #1,"test.dat","r"
        print chr$(peek(#1))," ";
        seek #1,2:print chr$(peek(#1))," ";
        seek #1,3,"here":print chr$(peek(#1))," ";
        seek #1,-2,"end":print chr$(peek(#1))," ";
        print tell(#1)
close #1
```

Das Programm erzeugt zuerst eine Datei test.dat, die den Text "abcdefghijkl" enthält. Danach werden einzelne Zeichen aus der Datei ausgelesen mit peek(). Zwischen den 'Peeks' wird die Position in der Datei durch seek geändert; schließlich ermittelt tell() die Position in der Datei. Der Ablauf des Programms erzeugt eine Ausgabezeile: "a c g k 11".
seek hat zwei oder drei Argumente. Die Dateinummer, die neue Startposition in der Datei und (wahlweise) die Position, von der aus die Startposition gezählt wird. Diese Position kann sein "begin" (Die Startposition zählt vom Anfang der Datei. Das ist die Vorgabe), "here" (Die Startposition zählt von der aktuellenPosition in der Datei) und "end" (Die Startposition zählt vom Ende der Datei).
Schließlich können Sie auch (ähnlich wie beim open Kommando) seek() in der einer if-Bedingung benutzen (z.B. if (!seek(...)) error "Shit !"); damit kann man auf einfache Weise den Erfolg eines seek() testen.
Unformatierten Text drucken
Sie können das Schlüsselwort printer ohne Anführungsstriche angeben anstelle eines Dateinamens, um auf Ihrem Drucker auszugeben. Hier ist ein Beispiel.
Anders herum ...
Jetzt, wo Sie das Beispielprogramm verstehen, sollten Sie sich diese neue Variante ansehen:

```
a=open("test.dat","r")
while(!eof(a))
   input #(a) a$
   print a$
wend
```

open() ist eine Funktion, um die erste freie Dateinummer zu ermitteln (vielleicht 1). Diese Dateinummer können Sie mit dem print-Befehl benutzen. Aber denken Sie daran, dass Yabasic runde Klammern nach der Raute erwartet (wie in input #(a) a$).

Import sub
Export sub
File$ = FILEPANEL Mode$, Title$, Directory$
Mode$ = "Load-File"
Mode$ = "Load-Directory"
Mode$ = "Load-File-and-Directory"
Mode$ = "Save-File"

Opens a filepanel in Directory$. According to the Mode$, you can either select a file,
a directory or both for loading or select/enter a file for saving. The selected file is File$.
File$ = FILEPANEL Mode$, Title$, Directory$, Filename$
Same as above, except that you can provide preset Filename$ for the mode "Save-File"

15. Systemnahe Operationen

Da YAB auf Yabasic basiert und nicht als Scriptsprache konzeptioniert worden ist, hat man dennoch eine Möglichkeit gesucht direkt mit dem System zu kommunizieren. Dazu gibt es den System-Befehl. Dieser ist in YAB zweimal vorhanden. Diese Unterscheiden sich in dem Dollarzeichen und die sich daraus resultierene Möglichkeit Rückgabe zu speichern und weiter zu verarbeiten.
Bei dem System Befehl übergibt man an einem exteren Programm, welches über den Terminal gestartet wird, seine Werte und diese können dann von den externen Programm weiterverarbeitet werden, oder man ruft über den System ein externes Programm und ließt dessen Ausgabe in eine Variable ein um diese Informationen in seinem YAB Programm weiterzuverarbeiten.

Als Beispiel dient dazu eigene Attribute in eine Datei schreiben und dann auslesen zu lassen.

Add_attribute$ = System$(addttr „Hallo test“)
Read_attribute = System$(readattr „Hallo test“)

System$

Man kann mit PEEK sehr viele Systemwerte auslesen, darunter fällt zum Beispiel die Position der Deskbar, wie breit diese ist und ob diese gerade ausgeklappt ist oder nicht. Desweiteren hat man die Möglichkeit, das Programmverzeichnis abzufragen, sowie die Breite der Scrollbars zu ermitteln. Diese und weitere Möglichkeiten werden in dem nachfolgenden Besipielen erklärt.

x = PEEK("DesktopWidth")	Gibt die Breite der Bildschirmauflösung wieder
y = PEEK("DesktopHeight")	Gibt die Höhe der Bildschirmauflösung wieder
x = PEEK("Deskbar-x")	Returns the position of the left side of the Deskbar
y = PEEK("Deskbar-y")	Returns the position of the top side of the Deskbar
w = PEEK("DeskbarWidth")	Gibt die Breite der Deskbar wieder
h = PEEK("DeskbarHeight")	Gibt die Höhe der Deskbar wieder
p = PEEK("Deskbarposition")	Gibt die Position der Deskbar im Uhrzeigersinn wieder, dabei bedeutet 1 = oben links 2 = oben 3 = oben rechts 4 = unten rechts 5 = unten 6 = unten links
i = PEEK("Deskbarexpanded")	Gibt True als Wert zurück, wenn die Deskbar ausgeklappt ist. Sollte dies nicht der Fall sein, wird FALSE zurückgegeben. Diese Funktion funktioniert nur wenn sich die Deskbar an der Position 1 oder 3 befindet.
Height = PEEK("menuheight")	Gibt die Höhe, in Abhängigkeit des eingestellten Fonts, der Menubar zurück, wenn das Fenster eine Menubar hat, ansonsten bekommt man den Wert -1 zurück.
Width = PEEK("scrollbarwidth")	Gibt die breite der Scrollbar zurück. Returns the width of the scrollbars. Different to "menuheight", it returns the correct width even if no scrollbars are used.
Height = PEEK("tabheight")	Gibt die Höhe des Reiters (Tab) zurück

Directory$ = PEEK$("directory")	Returns application directory. Use this in the following way:
	if (peek("isbound")) then
	Directory$ = peek$("directory")
	else
	Directory$ = system$("pwd")
	Endif

TrackerItem$ = PEEK$("refsreceived")	Returns TrackerItem which you used 'open with...' your application on.

Die PEEK zum gehörende Möglichkeit Dateien auslesen, wird in den Kapitel 14.Dateioperationen detailiert erklärt.

16. Drucken

PRINTER SETUP SetupFile$ Speichert die eingestellten Daten in die Datei SetupFile$.

PrintingError = PRINTER JobName$, SetupFile$, View$

Diese Zeile sorgt dafür, dass der View View$ mit dem Jobnamen JobName$ gedruckt wird. Dazu wird das SetupFile geladen, wenn diese angegeben wurde, ansonsten wird automatisch nachgefragt.

Hinweis: Alpha Transparenz wird nicht sauber gedruckt.
Hinweis: Due to a bug in BeOS and ZETA, there are probably problems printing CHECKBOX IMAGE and BUTTON IMAGE images, use DRAW IMAGE instead!

Folgende Rückgabewerte können beim Drucken auftreten:
0 = Keine Probleme
1 = Seiteneinstellungen sind fehlgeschlagen (Wahrscheinlich ist keine Kommunikation mit dem

Drucker möglich)
2 = Die Konfiguration wurde geladen, aber die Seiteneinstellungen sind Fehlgeschlagen
3 = Der View View$ wurde nicht gefunden
4 = Die Anzahl der Seiten ist <= 0
5 = Der Druckvorgang wurde aufgrund irgendeinem Problems abgebrochen

Hinweis: Wenn ein Fehler von 1 bis 4 auftritt, kann man davon ausgehen, das nichts zum Drucker übermittelt worden ist.
Bei Fehler 4 kann es daran liegen, das es ein Fehler im PDF Druckertreiber gibt. Dieser zeigt dann zwar eine Seitengröße an, obwohl gar keine ausgewählt ist.

99. Yab selbst erweitern

Tutorial zum Hinzufügen neuer Befehle in Yab

99.1 Einleitung

Yab basiert auf dem Yabasic Interpreter (http://www.yabasic.de). Auf der Yabasic Webseite finden Sie zusätzliche Quellen, wie man den Interpreter erweitert (Titel: "Guide into the Guts of Yabasic"). Dennoch soll dieses Dokument eine ausreichende Einleitung sein. Sie benötigen für das Hinzufügen neuer Befehle in yab grundlegendes Wissen über yab, C, C++ und der BeAPI. Wissen über Flex und Bison sind nicht zwingend notwendig.

99.2 Entwerfen eines Befehls

Ein typischer yab Befehl ist normalerweise entweder eine Prozedur (also eine Funktion die void zurückgibt), eine Funktion, die eine Zahl zurück gibt (double oder int) oder eine Funktion, die eine Zeichenkette (char*) zurückgibt.
Ein Befehl besteht aus

- dem Befehlswort (lexikalischer Einheit/token)
- der Befehlsregel (Grammatik)
- die gekapselte C-Funktion, die die C++ Funktion Methode aufruft
- die C++ Methode

Die Details hierzu werden ausführlich in den nächsten Abschnitten beschrieben.

99.2.1 Die Befehlswörter

Die benötigten Wörter für einen Befehl (tokens/Terminale) werden in der Datei yabasic.flex definiert. Kontrollieren Sie bitte vorher, ob es nicht schon ein Befehlswort gibt, was genau Ihren Befehl beschreibt. Somit werden unnötige doppelte Befehle vermieden.
Ein Befehlswort besteht aus dem Wort selbst (in Großbuchstaben) und aus dem Zeichen, das es zurückgeben soll. Das Zeichen ist häufig einfach das Wort selbst mit einem t davor.

Beispiel:

```
BUTTON return tBUTTON;
```

Anmerkung: Es gibt bei den Befehlswörtern Ausnahmen, z.B. tGet stellt das Befehlswort GET$ dar, während tGETNUM GET darstellt.

Neue Befehle (Tokens) müssen in der Datei yabasic.bison beschrieben werden. Dazu fügen Sie den Befehlsnamen am Anfang der Datei in der Befehlliste hinzu.

99.2.2 Die Befehlsregeln (Grammatik)

Die aktuelle Befehlsregeln (Grammatik) muss auch in der Datei yabasic.bison hinzugefügt werden. Wenn Sie sich die Datei anschauen, bekommen Sie eine gute Übersicht, wie die Grammatik aussehen soll.

Die folgenden drei Abschnitte sind interessant:

der Abschnitt für Prozeduren
der Abschnitt für numerische Funktionen

der Abschnitt für Funktionen, die Zeichenketten verarbeiten.
Wir beschreiben in den nächsten beiden Unterabschnitten die Unterschiede zwischen den Verfahren und den Funktionen.
Eine Befehlsregel wird am Anfang mit einem senkrechten Strich | (pipe) geschrieben, gefolgt von einem Zeichen und dem Befehlswort. Hinter dem Wort werden die Argumente geschrieben. Hinter dem letzten Argument folgt noch ein, in zwei geschweifte Klammern gefasster C Funktionsname z.B. {add_command(cBUTTON,NULL);} . Dieser Bezeichner wird mit einem Semikolon beendet.

Beispiel:

```
| tBUTTON coordinates to coordinates ',' string_expression ',' string_expression ',' string_expression
{add_command(cBUTTON,NULL);}
```

Es zwei Arten von Argumenten:

Das eine Argument ist eine expression, das ist immer eine Zahl von Typ double. oder
das Argument ist eine string_expression, also eine Zeichenkette von Type char*.

Sie können durch Kommata getrennt werden ",". Klammern ("(" und ")") sind auch möglich, aber ich habe sie so gut wie nie benutzt.
In diesem Beispiel hat der Befehl Button 7 Argumente, 4 Zahlen für die Koordinaten und 3 Zeichenketten. Die Zeichenketten enthalten die Identifikation des Buttons, den Buttontext und den View (auf dem der Button nachher liegen soll).

99.2.3 Prozeduren

Die Prozedur in dem obigen Beispiel gibt keinen Wert zurück, sondern es ist eine void-Funktion in C. Die Prozedur enthält einen internen Bezeichner, in dem obigen Beispiel cBUTTON. Dieser Bezeichner muss in der Datei yabasic.h beschrieben werden. Um den genauen Einfügepunkt für den Bezeichner zu finden, durchsuchen Sie die Datei nach ähnlichen Bezeichner und fügen Sie dann dementsprechenden an der Stelle den Bezeichner ein.

Außerdem muss in der Datei yabasic.h der Name der C-Funktion, die in der Datei graphic.c hinzugefügt wird, beschrieben werden. Prozeduren erhalten immer einen struct command *, YabInterface * als Argument. Der struct command enthält Informationen über die yab Argumente, Zeilennummer usw.
In graphic.c leitet die C-Funktion diese Informationen einfach an die C++ Klasse weiter (YabInterface).

Vor dem Hinzufügen der Funktion in graphic.c, müssen Sie die Funktion auch in der Datei main.c hinzufügen. Fügen Sie die Funktion der switch Abfrage hinzu, die entsprechend ihrem Bezeichner die Funktion benennt. z.B.:

```
case cBUTTON
   createbutton(current, yab); DONE;
```

Das folgende Beispiel zeigt eine typische Void Funktion in der Datei graphic.c:

```
void createbutton(struct command *cmd, YabInterface* yab)
{
  double x1,y1,x2,y2;
  char *id, *title, *view;

  view = pop(stSTRING)->pointer;
  title = pop(stSTRING)->pointer;
  id = pop(stSTRING)->pointer;
  y2=pop(stNUMBER)->value;
  x2=pop(stNUMBER)->value;
  y1=pop(stNUMBER)->value;
```

```
  x1=pop(stNUMBER)->value;

  yi_SetCurrentLineNumber(cmd->line, (const char*)cmd->lib->s, yab);
  yi_CreateButton(x1,y1,x2,y2, id, title, view, yab);
}
```

In unserem Beispiel zuerst werden die 7 yab Argumente vom Befehl struct zurückgeholt.

Anmerkung: Die Argumente werden auf einem Stack gelagert, also müssen Sie sie im umgekehrten Reihenfolge zurückholen! Hier wird z.B. y1 vor x1 zurückgeholt. Auch Zeichenketten und Zahlen haben unterschiedliche pop calls.

Zahlen können entweder von Typ double oder int sein, aber für Koordinaten sollte man immer double benutzten.

Die aktuelle Zeilennummer wird der YabInterface Klasse hinzugefügt, indem man yi_SetCurrentLineNumber aufruft. Diese Zeile ist für alle void Funktionen gleich.

Schließlich werden die Argumente an die YabInterface Klasse weitergeleitet, indem man yi_CreateButton aufruft.

99.2.4 Funktionen (Functions)

Funktionen, die entweder eine Zahl oder eine Zeichenkette zurückgeben, werden unterschiedlich eingefügt. Zunächst erhalten sie einen anderen Bezeichner. Dieser beginnt mit einem f z.B.. fLISTBOXGETNUM. Dieser Bezeichner muss in yabasic.h in den enum Funktionen beschrieben werden

Anmerkung: die Funktionen werden durch die Anzahl ihrer Argumente sortiert!

Wie schon zuvor, muss der Name der C-Funktion in die Datei graphic.c hinzugefügt und auch in yabasic.h beschrieben werden. Anders als bei den Prozeduren, bekommt die Funktion ihre Argument sofort, z.B.:

```
int listboxgetnum(const char*, YabInterface *yab, int line, const char* libname);
```

listboxgetnum gibt ein int bei einer Zeichenkette als Argument zurück. Die weiteren Argumente YabInterface *yab, int line, const char* libname müssen hinzugefügt werden, um die Informationen der YabInterface Klasse zur Verfügung zu stellen.

Anders als bei den Prozeduren sind die Argumente und der Funktionsaufruf in der Datei function.c gespeichert. Diese Argumente werden vom Stack zurückgeholt und an die Funktion in graphic.c weitergegeben.

Beispiel:

```
case fLISTBOXGETNUM:
  str=a1->pointer;
  value = listboxgetnum(str, yab, linenum, current->lib->s);
  result = stNUMBER;
break;
```

Das Zeichenkettenargument wird durch den a1->pointer zurückgegeben. Numerische Argumente werden durch z.B. a3->value adressiert (nicht in diesem Beispiel). Die Argumente werden von a1 bis a6 nummeriert. Mehr Argumente werden z.Z. nicht unterstützt.

Das Resultat wird als Zahl gespeichert. Das Resultat für Zeichenketten wird im stString (pointer and result = stSTRING;) gespeichert. Damit Sie eine Vorstellung bekommen, wie andere Befehle aussehen, schauen Sie sich die Datei an.

Schließlich muss die Wrapper-Funktion in graphic.c. eingefügt werden. Der folgende Code zeigt Ihnen wie es aussehen könnte:

```
int listboxgetnum(const char* id, YabInterface *yab, int line, const char* libname)
{
  yi_SetCurrentLineNumber(line, libname, yab);
  return yi_ListboxGetNum(id, yab);
}
```

Die Zeilennummer wird an die YabInterface Klasse weitergeleitet, indem man die Werte an die Funktion yi_SetCurrentLineNumber übergibt. Diese Zeile ist für alle Funktionen gleich. Die zurückgegebene Zahl wird dann einfach durch yi_ListboxgetNum weitergeleitet.

Anmerkung:Zeichenketten müssen mit my_strdup kopiert werden, z.B. return my_strdup ((char*) yi_CheckMessages (yab)); Bedenken Sie, dass die Zeichenketten noch im Speicher bestehen bleiben, wenn sie kopiert werden!

99.3 Die C++-Klasse YabInterface

99.3.1 Eine Methode hinzufügen

Nach den oben beschriebenen Punkten sind wir jetzt soweit eine neue Methode (Befehle) einzupflegen. Die Methode hat einen C++ Namen und eine Wrapper-Funktion mit einem externen Namen beginnend mit yi_. Beide müssen in YabInterface.h definiert und in YabInterface.cpp eingepflegt werden.

Die Wrapper-Funktion nimmt den Pointer zum YabInterface-Objekt und ruft die Hauptmethode auf:

```
void yi_CreateButton(double x1, double y1, double x2, double y2, const char* id, const char* title,
const char* view, YabInterface* yab)
{
  yab->CreateButton(BRect(x1,y1,x2,y2), id, _L(title), view);
}
```

Anmerkung: Das _L () Makro, das für alle Texte verwendet wird, übersetzt die Texte automatisch über das ZETA Local Kit. Es wird erst dann benutzt, wenn Sie den Befehl LOCALIZE verwenden. Bitte benutzen Sie dieses Makro so oft wie möglich, damit Sie eine einfache Lokalisierung der Programme ermöglichen.

Die Methode selbst ist ein Teil der YabInterface-Klasse, die von der BApplication-Klasse abgeleitet wird. Da der Interpreter in einem eigenen Thread läuft, sind alle BApplication-Methoden direkt von Ihrer Methode ansprechbar.

```
void YabInterface::CreateButton(BRect frame, const char* id, const char* title, const char* view)
{
  // code here
}
```

99.3.2 Zugriff auf yab-Datenstrukturen

Yab speichert verschiedene Informationen in Objektlisten. Die vermutlich wichtigste Liste ist die von den vorhandenen Views. Diese Liste ist in viewList gespeichert. Um ein neues Widget zu initialisieren, genügt es häufig das Elternteil der Views zu finden. Dieses erreichen Sie, indem Sie folgendes schreiben. YabList::GetView (const char*):

```
YabView *myView = cast_as((BView*)viewList->GetView(view), YabView);
if(myView)
{
```

```
   YabWindow *w = cast_as(myView->Window(), YabWindow);
   if(w)
   {
    w->Lock();
    // inizialize widget here
    w->unlock();
   }
   else   ErrorGen("Unalbe to lock window"); }
else
  Error(view, "VIEW");
```

Neue Widgets sollten automatisches Layout ermöglichen. Schauen Sie sich bitte die Befehle Button und Listbox an, damit Sie verstehen, was für unterschiedliche Arten von Layouting in yab verwendet werden.

Wenn Sie ein bestimmtes Widget auf einem Ihnen unbekannten View finden möchten, müssen Sie die Views durchsuchen. Solch eine Schleife sieht folgendermaßen aus (unter der Bedingung, dass MyWidget von BView abgeleitet wurde):

```
YabView *myView = NULL;
MyWidget *myWidget = NULL;
for(int i=0; iCountItems(); i++)
{
myView = cast_as((BView*)viewList->ItemAt(i), YabView);
  if(myView)
  {
   YabWindow *w = cast_as(myView->Window(), YabWindow);
   if(w)
   {
    w->Lock();
    myWidget = cast_as(myView->FindView(id), MyWidget);
    if(myWidget)
    {
     // do something with myWidget
     w->Unlock();
     return;
    }
   }
  }
 }
Error(id, "MyWidget");
```

Anmerkung: return wird nachdem etwas mit dem Widget gemacht worden ist und nachdem man wieder das Fenster geunlockt hat, aufgerufen. Dieses erlaubt eine einfache Fehlerüberprüfung am Ende der Schleife.

99.3.3 Kurze Anmerkungen

Am Ende einige Anmerkungen über

- BUILD-Makros: Einige Befehle haben BUILD-Makros, so dass man vollständige Teile yab beim Kompilieren abschalten kann. Das wird für die Buildfactory verwendet, um kleinere Codegröße zu produzieren. Nicht erforderliche oder unbenutzte Bibliotheken und/oder Codeteile werden einfach ausgelassen.
- Drawing: Drawing Befehle sind ein wenig trickreich. Sie erlauben das Zeichnen auf ein View über eine Bitmap und ein Canvas an. Besonders hat das View-Drawing ein eigenes Speichersystem. Deshalb schauen Sie sich die anderen Drawing Befehle an, um zu verstehen wie sie arbeiten.
- Eigene Klasse: Eigene Klassen hinzufügen ist schön, aber bedenken Sie auch, diese neuen Informationen in den Makefiles hinzuzufügen (R5 und ZETA!).

99.4 Zusammenfassung

Zum Schluß noch eine kleine Checkliste, damit Sie sehen können, ob Sie alle Punkte abgearbeitet haben.

- fügen Sie, wenn notwendig, das neue Befehlswort (token) in yabasic.flex hinzu
- fügen Sie die Befehlsregel (Grammatik) in yabasic.bison hinzu
- fügen Sie die Namen des Befehlidentifer und der C-Funktion in yabasic.h hinzu
- fügen Sie den Befehlsaufruf entweder in function.c oder in main.c hinzu
- fügen Sie Wrapper-Funktion in graphic.c hinzu
- fügen Sie die C und C++ Wrapper-Funktionen in YabInterface.h und in YabInterface.cpp hinzu
- fügen Sie die C++ Methode in YabInterface.h und in YabInterface.cpp hinzu

100. Kompendium

-= A =-

ALERT Text$, ButtonLabel$, Type$
Opens an alert window with Text$ and the button Label$ of type Type$
Type$ = "none/info/idea/warning/stop"
Selected = ALERT Text$, Button1$, Button2$, Button3$, Type$
A more general Alert window. Specify a text and up to three buttons.
If Button2$ or Button3$ is set to "", the button is not shown.
The selected button is returned, where Selected = 1 is the left, 2 the middle and 3 the right button.
Type$ = "none/info/idea/warning/stop"

-= B =-

BITMAP Width,Height, ID$
Creates a new bitmap in memory, you can draw on it
A bitmap is always initialized with a white transparent background
BITMAP GET x1, y1 to x2, y2, Target_bitmap$, Source_bitmap$
Copies specified area of Source_bitmap$ to Target_bitmap$.
BITMAP GET IconSize, Bitmap$, File$
Copies the icon, shown in Tracker, of File$ in the specified IconSize onto Bitmap$.
BITMAP GET ID$, Option$, Path$
copies an icon to a bitmap
An Option can be:
"Path" = Path$ must be the path to a file
"Mime" or "Mime16" = Path$ must be the mimetype of the file which icon is to be shown.
The resulting bitmap is 16 to 16.
"Mime32" = Path$ must be the mimetype of the file which icon is to be swown.
The resulting bitmap is 32 to 32.
BITMAP REMOVE Bitmap$
Removes Bitmap$ from memory
ErrCode = BITMAP SAVE Bitmap$, FileName$, Format$
Saves a bitmap as FileName$ (overwrites existing ones!) in a Format$ of one of the following:
{"png", "tiff", "ppm", "bmp", "jpeg", "tga"}
Note: that is the least common denominator for BeOS R5 PE and ZETA. For reasons of compatibility I can't support other translators (like e.g. gif).
ErrCode = 0 everything worked fine
ErrCode = 1 file could not be saved
BOXVIEW x1,y1 TO x2,y2, ID$, Label$, LineType, View$
Adds a box, note: the actual view of the BOXVIEW (where you can place your widgets) is at (x1+5,y1+10) to (x2-5,y2-5). This may be a subject of change!
LineType = 0 means no border
LineType = 1 means simple line border
LineType = 2 means fancy line border
BUTTON x1,y1 TO x2,y2, ID$, Label$, View$
Sets a button at position (x1,y1) to (x2,y2) with label Label$ on the view View$
BUTTON IMAGE x,y, ID$, EnabledPressed$, EnabledNormal$, Disabled$, View$
Create an image button at (x,y) on View$ with three files:
EnabledNormal$ The image of the released button
EnabledPressed$ The image of the pressed button
Disabled$ The image of the disabled button (you can put in an empty string "" if you don't need a disabled button)

-= C =-

CALENDAR x,y, ID$, Format$, Date$, View$
Open a calendar widget at (x,y), giving the format, the date and put that on the view View$.
Format$ = ("DDMMYYYY" or "MMDDYYYY") + ("." or "/" or "-"); default: "DDMMYYYY."
CALENDAR SET Calendar$, Date$
Set the date according to the current format (DDMMYYYY or MMDDYYYY).
date$ = CALENDAR GET$ Calendar$
Get the current date selected in Calendar$.
CANVAS x1,y1 to x2,y2, ID$, View$
A canvas is a special view for flicker-free drawing. It always shows a bitmap and thus cannot be resized. However, it does not have a drawing queue, so you do not need DRAW FLUSH.
A canvas is always initialized with a white background
CHECKBOX x1,y1, ID$, Label$, IsActivated, View$
Display a checkbox at (x1,y1) with Label$ on View$; if IsActivated is set to 0, the checkbox is off else it is on.
CHECKBOX SET CheckBox$, IsActivated
(De-)Activate the check box CheckBox$.
CHECKBOX IMAGE x,y, ID$, EnabledOn$, EnabledOff$, DisabledOn$, DisabledOff$, IsActivated, View$
Create an image checkbox at (x,y) on View$ with four files:
EnabledNormal$ The image of the released checkbox
EnabledPressed$ The image of the pressed checkbox
DisabledNormal$ The image of the normal disabled checkbox (you can put in an empty string "" if you don't need a disabled button)
DisabledPressed$ The image of the pressed disabled checkbox (you can put in an empty string "" if you don't need a disabled button)
Set isActivated to true/false when the checkbox should be on/off.
CLIPBOARD COPY Text$
Copy Text$ to the system clipboard.
Text$ = CLIPBOARD PASTE$
Paste ASCII text from the system clipboard into Text$.
COLORCONTROL x,y, ID$, View$
Create a color control widget at x,y. Note: it's width is 276 and it's height is 54 pixels.
COLORCONTROL SET ColorControl$, r,g,b
Set the color control ID$ to the color r,g,b.
Value = COLORCONTROL GET ColorControl$, "Red|Blue|Green"
Get the currently selected red/green/blue value of the colorcontrol.
COLUMNBOX x1,y1 TO x2,y2, ID$, HasHScrollbar, Option$, View$
HasHScrollbar is true, when the columnbox should get a horizontal scrollbar (it always has a vertical).
Option$ affects all columns! The columns can be made movable, resizable, removable
Option$ = "movable, resizable, popup, removable"
Option$ = "no-border" sets up the columnbox without border
COLUMNBOX COLUMN ColumnBox$, Name$, Position, MaxWidth, MinWidth, Width, Option$
Option$ = "align-left, align-center, align-right"
COLUMNBOX ADD ColumnBox$, Column, Row, Height, Item$
If Item$ = "__Icon__="+FileName$ then the image file FileName$ will be shown,
if Item$ = "__Path__="+FileName$ then the large Trackericon of the file FileName$ will be shown
If Item$ = "__Mime__="+Signature$ then the small icon of the mime type Signature$ will be shown
COLUMNBOX SELECT ColumnBox$, Row
Selects Row of ColumnBox$.
Row = 0 means deselect.
COLUMNBOX REMOVE ColumnBox$, Row
Removes the entries in Row of ColumnBox$
COLUMNBOX CLEAR ColumnBox$
Removes all entries of ColumnBox$.
COLUMNBOX COLOR ColumnBox$, Option$, r,g,b

Option$ = "selection-text, non-focus-selection, selection, text, row-divider, background"
n = COLUMNBOX COUNT ColumnBox$
Returns the number of entries in ColumnBox$.
Item$ = COLUMNBOX GET$ ColumnBox$, Column, Row
Returns the entry at Row in Column of Columbox$.

-= D =-

DRAW TEXT x1,y1, Text$, View$
Draws text at position (x1,y1) on View$
DRAW RECT x1,y1 TO x2,y2, View$
Draws a rectangle from (x1,y1) to (x2,y2) in the current high color on View$
DRAW BITMAP x,y, Bitmap$, Mode$, View$
Draw a bitmap on a view, another bitmap or a canvas.
Possible Mode$:
"Copy" -- copy the bitmap to the target ignoring transparency
"Alpha" -- copy the bitmap to the target supporting transparency
DRAW BITMAP x1,y1 TO x2,y2, Bitmap$, Mode$, View$
Draws and scales the bitmap Bitmap$.
If x2 is set to -1, the width is scaled according to the height;
if y2 is set to -1, the height is scaled according to the width;
if x2 and y2 are set to -1, the image is not scaled at all (this is like DRAW BITMAP x,y, Bitmap$, Mode$, View$).
Possible Mode$:
"Copy" -- copy the bitmap to the target ignoring transparency
"Alpha" -- copy the bitmap to the target supporting transparency
DRAW DOT x1,y1, View$
DRAW LINE x1,y1 TO x2,y2, View$
DRAW CIRCLE x1,y1, Radius, View$
DRAW CURVE x1,y1, x2,y2, x3,y3, x4,y4, View$
DRAW ELLIPSE x1,y1, xRadius, yRadius, View$
DRAW FLUSH View$
Deletes all former drawing commands from the drawing queue on View$
LoadError = DRAW IMAGE x,y, ImageFile$, View$
Draws the image ImageFile$ at position x,y on View$, returns LoadError
LoadError:
0 = success
1 = file not found
2 = translator roster not found
3 = translation failed
4 = ditaching bitmap failed
err = DRAW IMAGE x1,y1 TO x2,y2, Image$, View$
Draws and scales the image file Image$. Transparent images are set correctly now.
If x2 is set to -1, the width is scaled according to the height;
if y2 is set to -1, the height is scaled according to the width;
if x2 and y2 are set to -1, the image is not scaled at all (this is like DRAW IMAGE x1,y1, Image$, View$).
Width = DRAW GET "Text-Width", Text$, View$
Return the width in pixel of the string Text$ in the current font of View$.
Height = DRAW GET "Max-Text-Height", View$
Return the maximum height in pixel of the current font of View$. This is the size of the font plus how far characters can descend below the baseline.
FontFamilies$ = DRAW GET$ "FontFamily"
Returns a list of all installed fonts separated by "|".
FontStyles$ = DRAW GET$ FontFamily$
Returns a list of all font styles for a font FontFamily$ separated by "|".
DRAW SET FillOrStroke, Pattern$
FillOrStroke = 0 means fill
FillOrStroke = 1 means stroke

Pattern$ is a 8x8 1 bit map; one line is one character
Pattern$ = "HighSolidFill" means solid fill with the current high color
Pattern$ = "LowSolidFill" means solid fill with the current low color (default)
Pattern$ = "CheckeredFill" means checkered fill

DRAW SET Color$, r,g,b, View$
Color$ is one of the following:
"BGColor", r,g,b (216,216,216 default)
"HighColor", r,g,b (0,0,0 default)
"LowColor", r,g,b (216,216,216 default)

DRAW SET Color$, Option$, View$
Option$ is one of the following:
"Panel-Background-Color"
"Panel-Text-Color"
"Panel-Link-Color"
"Menu-Background-Color"
"Menu-Selection-Background-Color"
"Menu-Item-Text-Color"
"Menu-Selected-Item-Text-Color"
"Keyboard-Navigation-Color"
"Jan-Favorite-Color"
To those options the following can be added:
"Lighten-1-Tint"
"Lighten-2-Tint"
"Lighten-Max-Tint"
"Darken-1-Tint"
"Darken-2-Tint"
"Darken-3-Tint"
"Darken-4-Tint"
"Darken-Max-Tint"
"Disabled-Label-Tint"
"Disabled-Mark-Tint"
"Highlight-Background-Tint"

Example: draw set "bgcolor", "Menu-Background-Color, Lighten-1-Tint", View$

DRAW SET Font$, View$
Set the drawing font on View$.
Font$ = FontFamily + "," + FontStyle + "," + Size
or Font$ = "system-plain" for the plain system font,
or Font$ = "system-bold" for the bold system font,
or Font$ = "system-fixed" for the fixed system font

DRAW SET "Alpha", Alpha-Value
Sets the Alpha-Channel for the transparency of the drawing colors. Use this before setting the color with DRAW COLOR.
Alpha-Value is a value between 0 and 255 where 0 is total transparency and 255 is opaque.
Note: When Alpha-Value is below 255, use only HighSolidFill, patterns are ignored!
Note: Transparent drawing (that is Alpha-Value below 255) is not printed (see PRINTER) correctly!

DROPBOX x1,y1 TO x2,y2, ID$, Label$, View$
Adds an empty dropbox (BMenuField) at (x1,y1) to (x2,y2) known as ID$ and with a label on View$.

DROPBOX ADD DropBox$, Item$
Add Item$ to the dropbox. Use Item$ = "--" for a separator.

DROPBOX SELECT DropBox$, Position
Select the item at position Position. Counting starts at 0.

Item$ = DROPBOX GET$ DropBox$, Position
Get the item at Position.

n = DROPBOX COUNT DropBox$
Count the number of items.

DROPBOX CLEAR DropBox$
Clear the drop box from all items.

DROPBOX REMOVE DropBox$, Position
Removes item number Position.
Note: If the removed item was marked, the item before it will be marked instead. You will NOT get a message that the marked item has changed. If the first item will be deleted, the next item in the drop box will be marked.
Note: Each time an item was removed, the drop box count will change too!

-= F =-

File$ = FILEPANEL Mode$, Title$, Directory$
Mode$ = "Load-File"
Mode$ = "Load-Directory"
Mode$ = "Load-File-and-Directory"
Mode$ = "Save-File"
Opens a filepanel in Directory$. According to the Mode$, you can either select a file, a directory or both for loading or select/enter a file for saving. The selected file is File$.
File$ = FILEPANEL Mode$, Title$, Directory$, Filename$
Same as above, except that you can provide preset Filename$ for the mode "Save-File"

-= I =-

State = ISMOUSEIN(View$)
State = 0 the mouse cursor is not in the view View$
State = 1 the mouse cursor is in the view View$

-= K =-

Msg$ = KEYBOARD MESSAGE$(View$)
Works like INKEY$ on the command line (well, nearly; it does not wait for input as inkey$ does).

-= L =-

LAYOUT Layout$, WIndowOfView$
Set the layout for all views on the window of View$. The layout will affect all following new widgets, but not the already created. Draw commands are not affected by the layout, put them on an own view.
Layout$ (not case sensitive):
"Standard" = default layout, all widgets follow bottom and right side of the window
except for listboxes and textedit which follow all sides.
"All" = follow all sides (widgets resize)
"None" = follow the top and the left side (equals "top, left")
-OR-
Layout$ is a combination of a horizontal and a vertical command (e.g. "Right, Top" etc.).
Horizontal:
"Left" = follow left side (default, when no other horizontal layout is given)
"Right" = follow the right side
"Left, Right" = follow the left and the right side (resize)
"H-Center" = follow the horizontal center
Vertical:
"Top" = follow the top side (default, when no other vertical layout is given)
"Bottom" = follow the bottom side
"Top, Bottom" = follow the top and bottom side (resize)
"V-Center" = follow the vertical center
LISTBOX x1,y1 TO x2,y2, ID$, ScrollbarType, View$
Adds an empty listbox at (x1,y1) to (x2,y2) known as ID$ and with a ScrollbarType on View$.
ScrollbarType:
0 = none
1 = vertical scrollbars
2 = horizontal scrollbars

3 = vertical & horizontal scrollbars

LISTBOX CLEAR ListBox$

Removes all entries in ListBox$.

Entry$ = LISTBOX GET ListBox$, Row

Returns the Entry$ of Row in ListBox$.

LISTBOX REMOVE ListBox$, Item$

Removes an item Item$ from the Listbox ID$.

LISTBOX SORT Listbox$

Sort the entries of ListBox$ alphabetically.

-= L =-

n = LISTBOX COUNT ListBox$

Count the number of entries in the list box ListBox$

LISTBOX SELECT ListBox$, Position

Select the item at position Position.
Position = 0 means deselect

LISTBOX REMOVE ListBox$, Position

Remove the item at position Position.

LISTBOX ADD ListBox$, Item$

Add the item Item$ to the listbox; note: this replaces ITEM ADD, which is obsolete and will be removed.

LISTBOX ADD ListBox$, Position, Item$

Add the item Item$ at the position Position.

LOCALIZE

Enable automatic translation (Zeta only, otherwise nothing happens)

LOCALIZE FileName$

Enable automatic translation and use the locale file FileName$ (Zeta only, otherwise nothing happens)

-= M =-

MENU Head$, Menu$, Shortcut$, View$

Add a menu to the menubar with the menu head Head$, the menu item Menu$ and the shortcut Shortcut$ on View$.
Shortcut$ can contain the following modifiers at the beginning and the shortcut character at the end:
Modifiers$+ShortcutCharacter$
"S" for the shift key
"C" for the control key
“O" for the option key (on most keyboards probably the Windows button)
These modifiers can be combined, but the following combinations do not work:
"SO", "SC" and "SCO"
Note: The command key (ALT) is always part of the shortcut.

MENU SET MenuHead$, SetRadioMode, View$

Put the menu in radio mode, so up to one item is marked.
Note: you have to set the first marked item yourself.

MENU SET MenuHead$, MenuItem$, Option$, View$

Enable/Disable or mark/unmark or remove a menu item.
Option$ = "Disable|Enable|Mark|Plain|Remove"

msg$ = MESSAGE$

Collects the messages generated by the GUI elements.

Arrived = MESSAGE SEND Application$, Message$

Send a string to the yab application with signature Application$
(default signature is: "application/x-vnd.yab-app", you can change the signature when you add a comment in the first or second line of your program like this:
// mimetype "application/x-vnd.myapp"
Note: this does not work for bound programs currently!)
The destination yab application will produce a message$ stating:
_Scripting:...|

where the ... are the Message$.
The command returns one of the following error codes:
0: Message was delivered
1: The destination program was not found
2: The target's message queue is full
3: Time out, the message never made it to the target
4: An other error occurred

mouse$ = MOUSE MESSAGE$(View$)
Returns the state of the mouse related to View$.
It consists out of X:Y:LMB:MMB:RMB (where MB is the corresponding left, middle, right mousebutton).

-= M =-

mouse$ = MouseMove$
Returns messages generated by the mouse movement (for now works on Views,Buttons, PictureButtons, ColumnViews)
ControlName:_InsideView
ControlName:_EnteredView
ControlName:_ExitedView
ControlName:_MouseDown
ControlName:_MouseUp
ControlName:_LeftMouseButton
ControlName:_RightMouseButton
ControlName:_MiddleMouseButton
where ControlName = ID of control

MOUSE SET Option$
Hide or show the mouse cursor for this application.
Option$ = "Hide" -- hide the mouse cursor
Option$ = "Show" -- show the mouse cursor
Option$ = "Obscure" -- hide the mouse cursor until the mouse is moved

-= O =-

OPTION SET View$, "Auto-Resize"
Automatically resize the view View$ to preferred (when this is supported by the view).

OPTION SET View$, "Focus", HasFocus
Set/remove the focus of View$ (when this is supported by the view).

OPTION SET Control$, "Enabled", IsEnabled
Enable/disable the control Control$.

OPTION SET Control$, "Label", NewLabel$
Give a Control a new label.

OPTION SET Control$, "Visible", IsVisible
Set the control Control$ unvisible or visible
Note: The following widgets are controls:
CHECKBOX, RADIOBUTTON, SLIDER, TEXTCONTROL, BUTTON, COLORCONTROL, SPINCONTROL, CALENDAR

OPTION COLOR View$, "BGColor", r,g,b
Set the background color of any view (note: this is different to DRAW SET!)

-= P =-

x = PEEK("DesktopWidth")
Returns the current X-resolution of the current desktop

y = PEEK("DesktopHeight")
Returns the current Y-resolution of the current desktop

x = PEEK("Deskbar-x")
Returns the position of the left side of the Deskbar

y = PEEK("Deskbar-y")
Returns the position of the top side of the Deskbar

w = PEEK("DeskbarWidth")
Returns the height of the Deskbar
h = PEEK("DeskbarHeight")
Returns the width of the Deskbar
p = PEEK("Deskbarposition")
Returns the position of the Deskbar as follows (clockwise):
1 = top-left
2 = top
3 = top-right
4 = bottom-right
5 = bottom
6 = bottom-left
i = PEEK("Deskbarexpanded")
Returns true if the Deskbar is expanded (only possibe in position 1 and 3) and false if not.
Height = PEEK("menuheight")
Returns the height of the menu (which is related to the user's font settings).
This returns the height only when any window already has a menu, otherwise it returns -1.
Width = PEEK("scrollbarwidth")
Returns the width of the scrollbars.
Different to "menuheight", it returns the correct width even if no scrollbars are used.
Height = PEEK("tabheight")
Returns the height of the tabfield.
Directory$ = PEEK$("directory")
Returns application directory. Use this in the following way:

```
if (peek("isbound")) then
        Directory$ = peek$("directory")
else
        Directory$ = system$("pwd")
endif
```

TrackerItem$ = PEEK$("refsreceived")
Returns TrackerItem which you used 'open with...' your application on.
Selected$ = POPUPMENU x,y, MenuItems$, View$
Pop up a menu at position (x,y) on View$ with the menu items MenuItems$ which are separated by "|". Waits until an item is selected and returns it's label as Selected$.
PRINTER SETUP SetupFile$
Setup the printing enviroment and store it in file named SetupFile$.
PrintingError = PRINTER JobName$, SetupFile$, View$
Load the settings from SetupFile$ (or ask for them when the file is invalid) and print the view View$ as the job named JobName$.
Note: Alpha transparency is not printed correctly!
Note: Due to a bug in BeOS and ZETA, there are probably problems printing CHECKBOX IMAGE and BUTTON IMAGE images, use DRAW IMAGE instead!
Error codes for PrintingError:
0 = No Problems
1 = Page setup failed (probably communication problems with the print server)
2 = The configuration file was loaded but page setup failed
3 = The view View$ was not found
4 = The number of pages is 0 or less
5 = The printing was canceled or something went wrong with printing
Note: PrintingError = 4 can happen because of a bug in the PDF printing driver; althoug a page size is shown, none is selected. If this happens, you may want to call a PRINTER SETUP for the user and try to print again.
Note: When an error with code 1,2,3 or 4 occurs, you can be shure that nothing was sent to the printer yet.

-= R =-

RADIOBUTTON x1,y1, ID$, Label$, IsActivated, View$
Add a radio button at (x1,y1) and Label$ on View$. If isActivated is set to 0, the radio button is off else it is on. Radio buttons should be grouped together in one view.
RADIOBUTTON SET RadioButton$, IsActivated
(De-)Activate the radio button RadioButton$.

-= S =-

SCREENSHOT x1, y1 to x2, y2, Bitmap$
Takes a screenshot and copies the specified region of the desktop onto Bitmap$, which will be created!
SCROLLBAR ID$, ScrollbarType, View$
Make View$ scrollable.
SCROLLBAR SET Scrollbar$, "Vertical Position", Position
SCROLLBAR SET Scrollbar$, "Horizontal Position", Position
Set the scrollbar to the position Position. Default is (0,0).
SCROLLBAR SET Scrollbar$, "Vertical Range", Min, Max
-= S =-

SCROLLBAR SET Scrollbar$, "Horizontal Range", Min, Max
Set the scrollbars to a maximum range, default is (1000,1000).
Note: the Max value will be added to the view's actual width/height.
SCROLLBAR SET Scrollbar$, "Vertical Steps", SmallSteps, BigSteps
SCROLLBAR SET Scrollbar$, "Horizontal Steps", SmallSteps, BigSteps
Set the scrollbar steps.
SmallSteps are the scrolling steps when the user clicks on the arrow buttons (default is 1.0).
BigSteps are the scrolling steps when the user clicks on the empty part of the scrollbar (default is 10.0).
Position = SCROLLBAR GET Scrollbar$, "Horizontal"
Position = SCROLLBAR GET Scrollbar$, "Vertical"
Get the current position of the scrollbars.
SHORTCUT View$, Shortcut$, MyMessage$
Set a key Shortcut$ on View$, giving back the Message$ when used.
This is comparable to the shortcuts used in menus, but NOTE:
in opposite to a menu, you can't use command -X, -C, -V, -A, -W, -Q !
SLIDER x1,y1 TO x2,y2, ID$, Label$, Min, Max, View$
Add a slider with a minimum and and a maximum value (Min, Max).
SLIDER x1,y1 TO x2,y2, ID$, Label$, Min, Max, Option$, View$
Option$ = "block/triangle, horizontal/vertical"
SLIDER LABEL Slider$, StartLabel$, EndLabel$
Set start and end limit labels
SLIDER SET Silder$, Location$, Count
Set the hashmarks.
Note: The Zeta sliders do not distinguish between top/bottom/left/right marks, just use "both"
Location$ = "none/bottom/top/both" use hashmarks for horizontal sliders
Location$ = "none/left/rigth/both" use hashmarks for vertical sliders
Count = number of hashmarks
SLIDER COLOR Silder$, Part$, r,g,b
Part$ = "barcolor"
Part$ = "fillcolor"
SLIDER SET Silder$, Value
Set the slider to Value.
Value = SLIDER GET Slider$
Get the currently selected value of the slider.
ID = SOUND PLAY SoundFile$
Play the sound file SoundFile$ (can be anything BeOS/ZETA supports like wav, mp3, etc.).
You are given an ID to handle the playback later in your code.

SOUND STOP ID
Stop the sound with ID
SOUND WAIT ID
Waits until the sound with ID is finished playing.
SPINCONTROL x,y, ID$, Label$, Min, Max, Step, View$
Set a spin control for the range (Min,Max), counting by Step.
SPINCONTROL SET SpinControl$, Value
Set the spin control SpinControl$ to Value.
Value = SPINCONTROL GET SpinControl$
Get the current spin control value.
SPLITVIEW x1,y1 TO x2,y2, ID$, IsVerticalSplit, NormalStyle, View$
Set up the new view ID$ and split it into two new views ID$+"1" and ID$+"2".
If IsVerticalSplit = true, a vertical splitter is set,
If IsVerticalSplit = false, a horizontal splitter is set.
If NormalStyle = true, the splitter is 10 pixels thick,
If NormalStyle = false, the splitter is 4 pixels thick.
SPLITVIEW SET SplitView$ "Divider", Position
Set the position of the divider, default is the half of the total split view.
SPLITVIEW SET SplitView$ "MinimumSizes", LeftOrTop, RightOrBottom
Set the minimum sizes of the left (or top) view and the right (or bottom) view.
Position = SPLITVIEW GET SplitView$, "Divider"
Get the current position of the divider.
STACKVIEW x1,y1 TO x2,y2, ID$, NumberOfViews, View$
Set up a stack of views where only one is always visible.
STACKVIEW SET StackView$, ViewNumber
Set the view number ViewNumber as the visible view.
ViewNumber = STACKVIEW GET StackView$
Get the current number of the visible view.
STATUSBAR x1, y1 to x2, y2, ID$, Label1$, Label2$, View$
Creates a statusbar with ID$ and label(s) on View$.
Label1$ is on the left side above the actual bar and so is Label2$ on the right.
STATUSBAR SET ID$, Label1$, Label2$, State
Sets Statusbar ID$ to State and changes the Labels to specified ones.
SUBMENU MenuHead$, MenuItem$, SubMenuItem$, Modifier$, View$
This is the same as MENU, it just adds a submenu instead. Deeper leveling is not possible in yab, as a deeper menu structure would be bad style anyway.
SUBMENU SET MenuHead$, MenuItem$, SetRadioMode, View$
Put the submenu in radio mode, so up to one item is marked.
Note: you have to set the first marked item yourself.
SUBMENU SET MenuHead$, MenuItem$, SubMenuItem$, Option$, View$
Enable/Disable or mark/remove a submenu item.
Option$ = "Disable|Enable|Mark|Plain"

-= T =-

TABVIEW x1,y1 TO x2,y2, ID$, Option$, View$
The Option$ = "top, bottom" is ZETA only, R5 users have to set this too, but the R5-Tabs always stay on the top currently.
TABVIEW ADD TabView$, TabName$
For each tab, a new view is created; you can add widgets or draw on these views as ususal.
The ids for the views ist TabView$+"1" for the first, TabView$+"2" for the second view, etc.
TABVIEW SET TabView$, TabNumber
Open the tab number TabNumber.
TabNumber = TABVIEW GET TABVIEW$
Get the current opened tab.
TEXT x,y, ID$, Text$, View$
Displays Text$ at position (x,y) on View$. This cannot be flushed like DRAW TEXT and is meant for permanent labels.
TEXT x1,y1 TO x2,y2, ID$, Text$, View$
Displays Text$ on View$. This cannot be flushed like DRAW TEXT and is meant for

permanent labels.
Furthermore you can set the alignment for this command with the TEXT SET command.
TEXT SET Text$, Alignment$
Set the alignment of Text$ to either "align-left", "align-center" or "align-right".
TEXTCONTROL x1,y1 TO x2,y2, ID$, Label$, Text$, View$
Opens a textcontrol from (x1,y1) to (x2,y2) with Label$ and preset Text$ on View$
TEXTCONTROL SET TextControl$, Text$
Set the text control's text to Text$.
TEXTCONTROL SET TextControl$, IsPassword
IsPassword = false Normal typing
IsPassword = true Hide typing
TEXTCONTROL CLEAR TextControl$
Delete the text of the text control.
Text$ = TEXTCONTROL GET$ TextControl$
Get the entry of the text control TextControl$. Even works, when Return was not pressed.
TEXTEDIT x1,y1 TO x2,y2, ID$, ScrollbarType, View$
Opens an editor at (x1,y1) to (x2,y2) with ID$ (not displayed) on View$. For the scrollbartypes look at the listbox entry.
TEXTEDIT also follows all sides in standard layout.
TEXTEDIT ADD TextEdit$, Text$
Insert Text$ in the textedit TextEdit$ on View$.
TEXTEDIT CLEAR TextEdit$
Clears the text from the textedit TextEdit$ on View$.

-= T =-

EnteredText$ = TEXTEDIT GET$ TextEdit$
EnteredText$ holds the text of the textedit ID$ on View$
TextLine$ = TEXTEDIT GET$ TextEdit$, LineNumber
Width = TEXTEDIT GET TextEdit$, "Line-Width", LineNumber
Height = TEXTEDIT GET TextEdit$, "Line-Height", LineNumber
LineNumber = TEXTEDIT GET TextEdit$, Option$, Search$
LineNumber = Option$ = "Find", Search$
LineNumber = Option$ = "Case-Sensitive-Find", Search$
LineNumber = TEXTEDIT GET TextEdit$, Option$
IsChanged = Option$ = "hasChanged"
LineNumber = Option$ = "countlines"
LineNumber = Option$ = "currentline"
YOffset = Option$ = "vertical-scrollbar"
XOffset = Option$ = "horizontal-scrollbar"
Position = Option$ = "cursor-position"
TextLength = Option$ = "textlength"
TEXTEDIT SET TextEdit$, Option$
Applys an option to the textedit TextEdit$ on View$.
Option$ = "cut, copy, paste, clear, select-all, undo"
TEXTEDIT SET TextEdit$, Option$, Value
Option$ = "autoindent", true/false
Option$ = "wordwrap", true/false
Option$ = "editable", true/false
Option$ = "color-case-sensitive", true/false
Option$ = "changed", true/false
Option$ = "gotoline", LineNumber
Option$ = "select", LineNumber
Option$ = "tabwidth", TabWidth
Option$ = "textwidth", TextWidth
Option$ = "autocomplete-start"
Option$ = "has-autocompletion"
Default options are:
autoindent = false
wordwrap = true

editable = true
color-case-sensitive = false
changed = false
has-autocompletion = true
autocomplete-start = 4

TEXTEDIT SET TextEdit$, Option$, Value$
Option$ = "autocomplete"
Add a word Value$ to the auto-completion list.
Option$ = "font"
Set the font to Value$ (similar to DRAW SET); default is "system-plain"

TEXTEDIT COLOR TextEdit$, Option$, Command$
Option$ = "color1, color2, color3, color4, char-color", Command$
Add the command Command$ to the list of words that are checked for syntax highlighting
The char-color behaves differently and highlights only the first character of Command$
(this happens after the highlighting of the other colors).

TEXTEDIT COLOR TextEdit$, Option$, r,g,b
Option$ = "bgcolor, textcolor, color1, color2, color3, color4, char-color", r,g,b
Default colors are:
bgcolor = 255,255,255 (white)
textcolor = 0,0,0 (black)
color1 = 0,0,255 (blue)
color2 = 255,0,0 (red)
color3 = 0,250,0 (green)
color4 = 185,185,185 (gray)
char-color = 200,0,255 (magenta)

-= T =-

TEXTURL x,y, ID$, Label$, Address$, View$
Set the web/email/ftp address at (x,y) with the label Label$ and the url Address$.
This widget is quite nice, it launches the appropriate application when clicked, it supports
a right-click popup-menu and you can drag the url to create either a person file or a bookmark.
Email can be either of the style: "mailto:foo@bar.com" or just "foo@bar.com"
A web url starts either with "http://" or with "file://"
And a ftp address starts with "ftp://"

TEXTURL COLOR TextURL$, Option$, r,g,b
Set the colors for the URL. Valid options are:
"Label" for the color of the label,
"Click" for the color of the label, when clicked and
“Mouse-over" for the color of the label, when the mouse is moved over the label.

Id = THREAD GET Option$, Program$
Option$ = "TeamID" returns the team ID for the program named Program$
Specify the whole path with parameters (only the first 64 characters are important)
for the team, the first team with this path and parameters will be returned.
Option$ = "ThreadID" returns the thread ID for the program named Program$
Specify the program name, the first thread with this name will be returned.
Returns -1 when the program was not found.

Success = THREAD REMOVE Option$, ID
Option$ = "TeamID" kills the team with number ID
Option$ = "ThreadID" kills the thread with number ID
Returns True when successful and False otherwise.
Note: You can crash your system with this command! If you don't know what this command is
meant for, then don't use it!

TOOLTIP View$, Text$
Set Text$ as the tooltip information for any Widget or View View$.
Set Test$ = "" to remove the tooltip again.

TOOLTIP COLOR "bgcolor/textcolor", r,g,b
Set the background/text color of all tooltips to r,g,b. Note: This is BeOS R5 only!

Translation$ = TRANSLATE$(Source$)
Translation$ holds the translation of Source$ (Zeta only, returns Source$ otherwise)

TREEBOX x1,y1 to x2,y2, ID$, ScrollbarType, View$
Adds a tree box. This behaves just like a LISTBOX but is able to show nested trees. Note: ITEM ADD does not work here. Use TREEBOX ADD instead. ITEM ADD will be depricated in one of the next releases.
TREEBOX ADD TreeBox$, RootItem$
Add the item RootItem$ to the top level of the tree.
TREEBOX ADD TreeBox$, HeadItem$, ListItem$, IsExpanded
Add the item ListItem$ under the level of HeadItem$ of the tree.
TREEBOX CLEAR TreeBox$
Clear the tree.
TREEBOX REMOVE TreeBox$, ListItem$
Removes the first list item ListItem$ from the tree. Note: this also removes all subitems!
n = TREEBOX COUNT TreeBox$
Returns the number of entries in TreeBox$
Item$ = TREEBOX GET TreeBox$, Position
Returns the Item$ at position Position in TreeBox$.
TREEBOX EXPAND TreeBox$, Head$
Expands Head$ in TreeBox$.
TREEBOX COLLAPSE TreeBox$, Head$
Collapses Head$ in TreeBox$.
TREEBOX REMOVE TreeBox$, Position
Removes the entry at position Position in TreeBox$.
TREEBOX REMOVE TreeBox$, Head$, ListItem$
Removes ListItem$ under Head$ in TreeBox$.
TREEBOX SELECT TreeBox$, Position
Selects theitem at position Position in TreeBox$.
TREEBOX SORT TreeBox$
Sorts the entries of TreeBox$ alphabetically.

-= V =-

VIEW x1,y1 TO x2,y2, ID$, View$
Adds a view
VIEW DROPZONE View$
Define View$ as a drop zone that accepts dropped files.
DROPZONE now accepts multiple files (and sends them in inversed order as message)
VIEW REMOVE View$
Remove View$. The window view cannot be removed. Should be much more stable now.
Warning: Currently clears all internal information about menues and drop boxes. It is only safe To use it, when you don't have any menues or drop boxes on views that will not be removed.
Warning: Never remove menus with shortcuts, otherwise yab will crash when the shortcut key is pressed!
Result = VIEW GET View$, Option$
Option$ = "Position-X/Position-Y/Width/Height/Exists/Focused"
Returns the requested property or if used with Exists/Focused returns 1 if so and 0 if not.

-= W =-

WINDOW OPEN x1,y1 TO x2,y2, ID$, Title$
Open window at position x1,y1 to x2,y2 in screen coordinates with title Title$.
Automatically generates a window-sized view called ID$.
WINDOW CLOSE WindowView$
Closes window containing the view View$. Warning: this might destabilize your program if you try to reconstruct your old window or further use old views! The yab-taskforce is set on the tracks of this bug ;)
n = WINDOW COUNT
Returns the number of open windows
Result = WINDOW GET View$, Option$

Option$ = "Position-X/Position-Y/Width/Height/Minimum-Width/Minimum-Height/Maximum-Width/Maximum-Height/Exists"
Returns the requested property or if used with Exists returns 1 if found and 0 if not.
WINDOW SET Window$, Option$, Value$
"Look", "Document/Titled(default)/Floating/Modal/Bordered/No-Border"
"Feel", "Normal(default)/Modal-App/Modal-All/Floating-App/Floating-All"
See "BeBook->Interface Kit->BWindow->Constants and Defined Types" for details.
"Title", Title$
"Flags", "Not-Closable, Not-Zoomable, Not-Minimizable, Not-H-Resizable, Not-V-Resizable, Not-Resizable, No-Workspace-Activation, Accept-First-Click"
See "BeBook->Interface Kit->BWindow->Constants and Defined Types" for details.
"Flags", "Reset"
Resets the flags back to none.
"Workspace", "All"
Causes the window to appear on all workspaces.
"Workspace", "Current"
Causes the window to appear on only the current workspace.
WINDOW SET Window$, Option$, r,g,b
"BGColor", r,g,b (216,216,216 default)
"HighColor", r,g,b (0,0,0 default)
"LowColor", r,g,b (216,216,216 default)
WINDOW SET Window$, Option$, x,y
"ResizeTo", x,y
"MoveTo", x,y
"MinimumTo", x,y
"MaximumTo", x,y

-= W =-

WINDOW SET WindowView$, Option$
Option$ = "Activate"
Activate the window, so it is in the foreground.
Option$ = "Deactivate"
Deactivate the window, so it is in the background.
Option$ = "Minimize"
Minimize the window, or restore the window if it is already minimized.
Option$ = "Maximize"
Maximize (zoom) the window.
Option$ = "Enable-Updates"
Updates the window again after a "Disable-Updates".
Option$ = "Disable-Updates"
Disables the automatic window updates.

101. Reservierte Wörter in YAB

Von Yabasic benutzte Ausdrücke, die nicht als beliebige Variable benutzt werden sollten.

argument		elseif		local		restore	until
abs		elsif		log		return	up
acos		end		loop		reverse	upper$
and		enter		lower$		right$	using
arraydimension		eof		ltrim$		rightscreen	val
arraysize		eor		max		rinstr	version
as		error		mid$		rtrim$	wait
asc		esc		min		scrdown	warning
at		euler		mod		screen	wend
atan		execute		mouseB		screenheight	while
backspace		exit		mousebutton		screenwidth	window
beep		exp		mousemod		scrup	writing
bell		export		mousemodifier		seek	xor
bind		f1...f12		mousex		sig	
bitblit		fatal		mouseY		sin	
bitblit$		fi		new curve		sleep	
bitblt		filled		next		split	
box		fontheight		note		split$	
break		for		numparams		sqr	
case		frac		on		sqrt	
cb		getbit$		open		static	
cc		glob		origin		step	
chr$		gosub		os		str$	
circle		goto		pause		sub	
clear		hex$		peek		subroutine	
close		home		peek$		switch	
color		infolevel		pi		system	
colour		inkey$		poke		system$	
continue		input		printer		tab	
cos		ins		putbit		tan	
curve		instr		putscreen		tell	
data		interrupt		Ran		textalign	
date$		label		rc		then	
debug		left		read		time$	
dec		left$		reading		to	
default		len		rect		token	
del		let		rectangle		token$	
dim		library		redim		triangle	
down		line		repeat		trim$	

102. ASCII Tabelle

ASCII				ASCII			ASCII			ASCII		
hex	dez	Zeichen		hex	dez	Zeichen	hex	dez	Zeichen	hex	dez	Zeichen
0	0	NUL	^@	20	32	SP	40	64	@	60	96	`
1	1	SOH	^A	21	33	!	41	65	A	61	97	a
2	2	STX	^B	22	34	"	42	66	B	62	98	b
3	3	ETX	^C	23	35	#	43	67	C	63	99	c
4	4	EOT	^D	24	36	$	44	68	D	64	100	d
5	5	ENQ	^E	25	37	%	45	69	E	65	101	e
6	6	ACK	^F	26	38	&	46	70	F	66	102	f
7	7	BEL	^G	27	39	'	47	71	G	67	103	g
8	8	BS	^H	28	40	(	48	72	H	68	104	h
9	9	TAB	^I	29	41	)	49	73	I	69	105	i
0A	10	LF	^J	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	^K	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	^L	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	^M	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	^N	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	^O	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	^P	30	48	0	50	80	P	70	112	p
11	17	DC1	^Q	31	49	1	51	81	Q	71	113	q
12	18	DC2	^R	32	50	2	52	82	R	72	114	r
13	19	DC3	^S	33	51	3	53	83	S	73	115	s
14	20	DC4	^T	34	52	4	54	84	T	74	116	t
15	21	NAK	^U	35	53	5	55	85	U	75	117	u
16	22	SYN	^V	36	54	6	56	86	V	76	118	v
17	23	ETB	^W	37	55	7	57	87	W	77	119	w
18	24	CAN	^X	38	56	8	58	88	X	78	120	x
19	25	EM	^Y	39	57	9	59	89	Y	79	121	y
1A	26	SUB	^Z	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	^[	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	^\	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	^]	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	^^	3E	62	>	5E	94	^	7E	126	~
1F	31	US	^_	3F	63	?	5F	95	_	7F	127	DEL

103. HTML Code Tabelle

HTML-	ISO-8859-1				HTML-	ISO-8859-1				HTML-	ISO-8859-1		
code	hex	dez	Zch.		code	hex	dez	Zch.		code	hex	dez	Zch.
	A0	160			À	C0	192	À		à	E0	224	à
¡	A1	161	¡		Á	C1	193	Á		á	E1	225	á
¢	A2	162	¢		Â	C2	194	Â		â	E2	226	â
£	A3	163	£		Ã	C3	195	Ã		ã	E3	227	ã
¤	A4	164	¤		Ä	C4	196	Ä		ä	E4	228	ä
¥	A5	165	¥		Å	C5	197	Å		å	E5	229	å
¦	A6	166	¦		Æ	C6	198	Æ		æ	E6	230	æ
§	A7	167	§		Ç	C7	199	Ç		ç	E7	231	ç
¨	A8	168	¨		È	C8	200	È		è	E8	232	è
©	A9	169	©		É	C9	201	É		é	E9	233	é
ª	AA	170	ª		Ê	CA	202	Ê		ê	EA	234	ê
«	AB	171	«		Ë	CB	203	Ë		ë	EB	235	ë
¬	AC	172	¬		Ì	CC	204	Ì		ì	EC	236	ì
	AD	173			Í	CD	205	Í		í	ED	237	í
®	AE	174	®		Î	CE	206	Î		î	EE	238	î
¯	AF	175	¯		Ï	CF	207	Ï		ï	EF	239	ï
°	B0	176	°		Ð	D0	208	Đ		ð	F0	240	ð
±	B1	177	±		Ñ	D1	209	Ñ		ñ	F1	241	ñ
²	B2	178	²		Ò	D2	210	Ò		ò	F2	242	ò
³	B3	179	³		Ó	D3	211	Ó		ó	F3	243	ó
´	B4	180	´		Ô	D4	212	Ô		ô	F4	244	ô
µ	B5	181	µ		Õ	D5	213	Õ		õ	F5	245	õ
¶	B6	182	¶		Ö	D6	214	Ö		ö	F6	246	ö
·	B7	183	·		×	D7	215	×		÷	F7	247	÷
¸	B8	184	¸		Ø	D8	216	Ø		ø	F8	248	ø
¹	B9	185	¹		Ù	D9	217	Ù		ù	F9	249	ù
º	BA	186	º		Ú	DA	218	Ú		ú	FA	250	ú
»	BB	187	»		Û	DB	219	Û		û	FB	251	û
¼	BC	188	¼		Ü	DC	220	Ü		ü	FC	252	ü
½	BD	189	½		Ý	DD	221	Ý		ý	FD	253	ý
¾	BE	190	¾		Þ	DE	222	Þ		þ	FE	254	þ
¿	BF	191	¿		ß	DF	223	ß		ÿ	FF	255	ÿ

105. Revisionübersicht