Handbuch für ExpertItems

Von Werner Gutdeutsch

Inhalt

Handbuch für ExpertItems 1

I. ExpertItems 2

II. Das Bild: drei Felder, der Dialog, der Ausdruck „root“ 2

III. Items 3

IV. Die Eingabefelder 6

V. Programmieren mit Ja/Nein-Items (boolesche Items) und Aufzählungen (Menüeingaben) 9

VI. Programmieren mit Text-Items 18

VII. Programmierung mit Items eines quantitativen Typs 21

VIII. Programmieren mit Items des Typs Datum 23

IX. Hilfen zur Übersichtlichkeit 25

X. Ordnung von Items durch Listen und Einzelknoten 27

XI. Weitere Verwendung der Listen (Menüpunkt other) 31

XII. Spezielle Additionskolonnen 33

XIII. Mehrspaltig rechnen (Vektoren) 35

XIV. Funktionen und Variablen 35

XV. Transparenzfunktionen 37

XVI. Programmieren mit Klassen (Classes) 39

XVII. Kopieren von Items, Knoten und Klassen 40

Anhang: 40

I. Zur Handhabung von ExpertItems 40

II. Fehlerbehandlung: 41

III. Programmierratschläge: 45

Weiterfragen nur wenn nötig 45

Programmieren: unnötige Meldungen vermeiden 46

IV. Besondere Programmierprobleme 46

Versuche 46

V. Programmierbeispiele: 46

Index 55

I. ExpertItems

ExpertItems ist ein Dialogprogramm, welches jedem Experten ermöglichen soll, sein theoretisches Fachwissen in Form von logischen und mathematischen Zusammenhängen mit Hilfe von **Items** in ein Dialogprogramm umzusetzen.

Das erzeugte Dialogprogramm soll für den Letztanwender so transparent wie möglich sein. Auf Wunsch kann er alle Zwischenschritte der Berechnung einsehen.

Die logischen und mathematischen Zusammenhänge, die nötigen Fragen des zu erzeugenden Programms und die nötigen Meldungen werden von ExpertItems im Dialog abgefragt. Daraus erzeugt ExpertItems automatisch ein Anwenderprogramm.

Das hat zur Folge, dass der Experte beim Programmieren sein Augenmerk anfangs nicht auf die Dialogfolge richten darf, denn das ist Sache des Programms. Vielmehr muss er versuchen, sich aller logisch mathematischen Zusammenhänge bewusst zu werden. Das geschieht am besten dadurch, dass er eine Stoffsammlung fertigt, welche alle bekannten logisch-mathematischen Zusammenhänge enthält, und diese dann abarbeitet, indem er sie in Items umsetzt. Dabei wird dann auch sichtbar, welche Zusammenhänge etwa noch zu bestimmen sind. Dann lässt man das Anwender-Programm entstehen.

Als Nächstes sollte man sich bewusst sein, dass für jedes Item vier Einträge von zentraler Bedeutung sind, nämlich der Name, der das Item überhaupt erst erzeugt, der Typ (am Anfang), die Frage und der Rechenausdruck. Über die anderen Fragen kann man zu Beginn immer hinwegsehen.

Die Eingabefolgen dieses gänzlich automatisch erzeugten Programms können allerdings von den Erwartungen des Letztanwenders abweichen. Der Experte kann diese automatisch erzeugte Reihenfolge im Programmablauf jedoch verändern und den Erwartungen anpassen und sollte das selbstverständlich auch tun. Für diese Eingriffe stehen nur soweit unvermeidlich eigene Fragefelder zur Verfügung, denn hier beginnt - auf niedrigster Ebene – eine eigenständige Programmierung.

ExpertItems vermeidet so, dass der Experte mit einem Programmierer zusammenarbeiten muss, wenn er sein Fachwissen in ein Beratungsprogramm einbringt: Die klassische Zusammenarbeit verläuft ja so, dass der Experte zuerst ein Pflichtenheft schreibt, das dem Programmierer sagt, was das Programm leisten soll. Der Programmierer führt die Anweisung aus und der Experte prüft danach, ob ihm das Ergebnis gefällt. Wenn nicht, schlägt er Änderungen vor u.s.w. Da beide unterschiedlich vorgebildet sind, treten häufig Missverständnisse auf. Dieser Kommunikationsprozess wird eingespart, wenn der Experte in die Lage versetzt wird, das Programm selbst zu erzeugen und sogleich auszuprobieren. Eben das ist das Ziel von ExpertItems: Es soll den Programmierer ersetzen.

II. Das Bild: drei Felder, der Dialog, der Ausdruck „root“

1. Das Programmfenster (das beliebig vergrößert oder verkleinert werden kann) ist in drei Felder aufgeteilt: rechts befindet sich das Arbeitsblatt, in dem (im erzeugten Anwenderprogramm) gerechnet wird und in dem ExpertItems die Fragen stellt, die der Experte zur Erzeugung des Programm beantworten muss. Auf der linken Seite oben befindet ein „treeview“, das Orientierungsbäumchen, das alle Items anzeigt, sodass man durch Anklicken dort hinspringen kann.

Außerdem können auf diese Weise Unterverzeichnisse zusammengeklappt und wieder geöffnet werden, was die Übersicht erleichtert. Auf der linken Seite unten befindet sich das „Hilfefeld“ mit Erläuterungen zu dem Eingabefeld (der Frage), welche auf dem Arbeitsblatt rechts aktiv ist.

2. ExpertItems fragt, der programmierende Experte antwortet. Den meisten Eingabefeldern sind Fragen zugeordnet, die zu beantworten sind. Nur die Frage nach dem Item-Typ (s.u. III.4.) beschränkt sich darauf, in einem Menü die möglichen Typen zur Auswahl anzubieten. Sonst erwarten die Fragen von ExpertItems an den Anwender, dass Texte oder Rechenausdrücke eingegeben werden. Die meisten dieser Fragen welche ExpertItems dem Programmierenden stellt, können mehrzeilig eingegeben werden. Das zeigt sich darin, dass die Frage von ExpertItems an den Anwender nicht links neben dem Antwortfeld, sondern darüber steht. Die nächste Zeile im gleichen Feld wird erreicht, indem man nicht <Enter> eingibt (dann kommt man in das nächste Feld), sondern <Strg Enter>: Das fügt dem Eingabefeld eine weitere Zeile hinzu und bricht ggf. die Zeile, welcher sich die Schreibmarke befindet um.

3. Die erste Frage betrifft den **Expertenmodus**, welcher einem erfahrenen Programmierer den Umgang mit den Klassen von C++ eröffnet. Wird die vorgeschlagene Antwort <nein> bestätigt, verbleibt die Überschrift „r**oot**“. Das ist der Standardname der einzigen Klasse, welche im vereinfachten Modus alle programmierten Items umfasst. Sie werden dieser Klasse zugeordnet. Deshalb lautet die Anfangsfrage nach ihrem Namen „Name d. 1. Items d. root“, weil nämlich die grundlegende Klasse „root“ (d.h. Wurzel) heißt. Für eine isolierte Programmierung mit ExpertItems genügt dieser Standardname. Soll ein so erzeugtes Programm dem Letztanwender zur Verfügung gestellt werden, ist noch die Einbindung in WinFam oder ein ähnliches Programm oder die Ausgestaltung zu einem selbstständigen Programm zur Verwendung auf einem lokalen Rechner (wie WinFam) oder im Internet (wie ifam) erforderlich. Bitte nehmen Sie dazu Kontakt auf zu Dr. Ulrich Gutdeutsch, Am Egernfeld 5, D-85748 Garching bei München tel. 089 326 2238.

III. Items

Items sind die Bausteine von ExpertItems. Sie bilden ein Netz von gegenseitigen Verweisungen und halten mit der „Außenwelt“ Kontakt durch Fragen an den Letztanwender und Meldungen an ihn. Ein Item hat die Aufgabe, seinen eigenen Wert zu bestimmen. Dazu dienen seine Bestimmungsfunktion bzw. seine Bestimmungsfunktionen, welche den Wert entweder mit Hilfe eines Rechenausdrucks von dem Wert anderer Items herleitet, oder vom Endanwender abfragen. Die Wertbestimmung findet immer dann statt, wenn entweder der Anwender oder ein anderes Item, welches seinen Wert bestimmen soll, danach „fragt“. Der Wert (z.B. ein Unterhaltsbetrag, das Datum des Rentenbeginns eines Arbeitnehmers oder ein Haftungsanteil in Prozent) wird im Programmlauf (dem Anwenderprogramm, das von ExpertItems erzeugt wird) nur einmal bestimmt, und nur dann, wenn das für das gewünschte Endergebnis nötig ist. Die Bestimmung des Itemwerts erfolgt

entweder durch Frage an den Endanwender

oder

er errechnet sich aus den Werten anderer Items.

(so wie sich z.B. der Unterhalt aus Alter und Einkommen der Beteiligten errechnet).

Jedes Item bekommt deshalb:

1. Einen Namen,

mit dem es aufgerufen wird, wenn es von einem anderen Item zur Bestimmung von dessen Wert verwendet wird

Für den Aufruf genügt allerdings oft nicht der Name. Vielmehr muss manchmal zusätzlich noch angegeben werden, wie das aufgerufene Item vom aufrufenden aus zu finden ist, Einzelheiten hierzu s.u. IX. Ordnung von Items durch **Knoten**

Durch die Eingabe des Itemnamens in das dafür geöffnete Eingabefeld wird das Item angelegt mit der Folge, dass sich unterhalb dieses neu angelegten Items sich erneut ein Fragefeld für die Anlage eines weiteren Items öffnet.

2. Einfügen, Löschen und Verschieben eines Itemnamens

Man kann ein neues Item in eine Reihe vorhandener Items einfügen, indem man entweder in den Namens des Items, vor der das neue Item eingefügt werden soll mit der rechten Maustaste „Listenelement einfügen“ wählt oder in den Namen des Items hineinklickt und im Hängemenü die Option „Listenelement einfügen“ wählt. Es verschwinden dann die folgenden Items solange, bis der Name des neuen Items eingegeben ist. Man kann Items löschen. Wenn man nur das Eingabefeld löscht, dann verschwinden alle darunter angelegten Items. Jedoch ist der alte Itemname im Eingabefeld noch zu finden, indem man zuerst in das Eingabefeld für den Itemnamen und danach auf den dort erscheinenden Pfeil im Eingabefeld rechts klickt. Wird der Name wieder eingefügt, erscheinen auch alle folgenden Items wieder.

Will man den Itemnamen (und damit das Item) beseitigen, ohne die darunter befindlichen Items zu beeinträchtigen, so muss man die Löschfunktion verwenden. Entweder man klickt in das Namensfeld des Items und sucht im Hängemenü **Bearbeiten/Listenelement löschen**, oder man geht mit der Maus auf das Namensfeld ohne hinein zu klicken und findet mit der rechten Maustaste die Option **Listenelement löschen.**

Wird das Item in dieser Weise gelöscht, ist es auch nicht verloren. Vielmehr taucht es auf wenn an anderer Stelle entweder mit dem Hängemenü oder mit der rechten Maustaste **Listenelement einfügen** gewählt wird. Die Löschfunktion dient deshalb auch dazu, die Items zu verschieben.

Diese Verschiebemöglichkeit beschränkt sich allerdings auf die Items in derselben Reihe. Die Unter-Items eines **Knotens** werden so nicht erreicht. Nur der Gruppierungsknoten schaffen keine echten Unter-Items und ändern deshalb nichts an der Erreichbarkeit für die Verschiebung.

Bei der Einfügung eines neuen Items erscheint somit regelmäßig das zuletzt gelöschte Item. Das kann dann einfach überschrieben werden.

Man kann das Erscheinen des alten Items aber auch verhindern, indem man mit der Funktion **Optionen/Funktionen nur in Betaversion/Alteingaben löschen** alle Alteingaben beseitigt. Ist

dadurch kein gelöschtes Item vorhanden, das an die Stelle des gelöschten treten kann, dann verschwinden die folgenden Items solange, bis der neue Itemname eingegeben wird. [1]

3. Bestimmungsfunktionen,

welche den Wert dieses Items entweder direkt erfragen oder mit Hilfe der Werte anderer Items ermitteln, und

4. Einen Typ,

wobei ExpertItems eine Vielzahl von Typen bietet, was die Erstellung jeglicher textorientierter Programme ermöglicht.

Itemtypen:

a) Einfache Typen (s.u. IV-VI)

z.B. **Ja/Nein**, wenn nur die Werte **wahr** oder **falsch** möglich sind, **Aufzählung** für Menüeingaben, **Ganzzahl** für ganze Zahlen und **Kommazahl** für Dezimalbrüche und sonstige rationale Zahlen wie *0,15* oder *1/7*, **Geldbetrag** für Geldbeträge, **Text (**ggf. **mehrzeilig**) für Texte und **Datum** für Kalenderdaten.

In das Bausteinschema der Items ordnen sich auch weitere Typen ein, welche sich vom Grundtyp mehr oder weniger stark unterscheiden:

b) komplexe Typen (s.u. VI, XI),

nämlich **Datum** für Kalenderdaten (die ja aus mehreren Zahlen bestehen), sowie **mehrspaltige Kommazahl, mehrspaltiger Geldbetrag** und **mehrspaltiger Text** für mehrspaltiges Rechnen. Alle Items sind insofern funktional, als sie auf andere Items referenzieren können. Doch kennt ExpertItems auch

c) Itemtypen mit einer jeweils ganz speziellen Funktionalität (s.u. VII, IX, X),

die wiederum je nach dem Typ ihres Ergebnisses einem einfachen oder einem komplexen Typ zugehören können. Das sind **Kolonne von Geldbeträgen** und **eingerückte Kolonne von Geldbeträgen** für Additionskolonnen sowie **Listenelemente zählen, Listenelemente addieren** und **Kumulative Addition** welche auf **Listen** referenzieren. **Listen** sind als solche keine echten Items, sondern gehören zu den Strukturitems (s.u. (7)).

[1] Die Besonderheiten bei Löschen, Einfügen und Verschieben sind technisch bedingt. Es soll in Zukunft die Funktionalität von copy-and-paste übernommen werden. Angesichts des erheblichen Aufwands lässt sich noch nicht sagen, bis wann hier eine technische Verbesserung möglich sein wird.

Neben den bisher dargestellten **echten Items** verwendet ExpertItems für besondere Aufgaben auch

d) unechte Itemtypen,

nämlich **Funktion** und **Variable (s.u. XII),** welche anders als echte Items nacheinander unterschiedliche Werte annehmen können. Unecht sind aber auch Items, denen ein Ergebniswert überhaupt fehlt, nämlich die

e) Strukturitemtypen

welche andere Items ordnen. Das sind: **Gruppierungsknoten, Einzelknoten, Namensliste, Nummerierte Liste, Zugeordnete Liste (s.u. VII, VIII).**

Diese verschiedenen Itemtypen machen ExpertItems zu einem sehr mächtigen Tool, welches aber auf einer sehr einfachen Basis aufbaut. Im Folgenden soll die Anwendung von ExpertItems vom Einfachen bis zu immer komplexeren Leistungen dargestellt werden.

5. Ein zu bestimmendes Item

Die Zusammenhänge allein bewirken noch nicht den Programmlauf. Das Programm benötigt hierzu auch noch die Information, welche Item-Werte bestimmt werden sollen. Hierzu ist am Ende des Programms in das Feld „Zu bestimmendes Item (oder Startfunktion)“ der Name des Items einzutragen, dessen Wert auf jeden Fall, also nicht abhängig von vorherigen Eingaben, bestimmt werden soll. Das dort eingetragene Item ruft dann weitere Items auf, dessen Wert es für die Wertbestimmung benötigt. Fehlt der Eintrag, dann kompiliert das Programm zwar, stellt aber keine der programmierten Fragen.

Es können auch mehrere Items hier eingetragen werden. Beim jetzigen Programmstand muss dann aber die Funktionalität von C/C++ dafür in Anspruch genommen werden: Die Items müssen als Funktionen mit Klammern und Semikolon eingegeben werden, z.B. „Unterhalt_des_Kindes(); Unterhalt_des_Ehegatten();“.

IV. Die Eingabefelder

Die Items haben einen regelmäßigen Aufbau mit überwiegend gleichen Fragen:

1. **Itemtyp**:
Hier wird keine förmliche Frage gestellt, sondern die Auswahl eines Item-Typs in einem zwei-stufigen Menüsystem angeboten.

2. **Überschrift vor der Bestimmung des Itemwerts**:
Hier kann eine Überschrift eingetragen werden, welche vor die weiteren Meldungen des Items oder der von ihm aufgerufenen Items gesetzt wird. Die Überschrift muss in Anführungszeichen gesetzt werden. Doch kann dieser Text auch durch den Namen eines Text-Items – das dann ohne Anführungszeichen einzutragen ist – bestimmt werden.

Wenn auf die Überschrift keine weiteren Texte folgen (etwa weil eine Null-Eingabe zur Platzersparnis nicht ausgeben wird), verschwindet auch die Überschrift, weil sie dann sinnlos ist. Man kann die Ausgabe einer Überschrift aber auch durch Nutzung der C/C++-Funktion erzwingen:[2]
Nur wenn **Überschrift vor der Bestimmung des Itemwerts eingetragen wurde, öffnet sich ein** Fenster mit der Frage

3. Besondere Überschrift für das Übersichtsbäumchen:
Das Programm schlägt die vorher eingegebene Überschrift vor. Man kann aber hier einen anderen – kürzeren – Text eingeben, der sich für das Übersichtsbäumchen links oben besser eignet. (Die eingetragenen Überschriften erscheinen später beim Programmlauf links oben im Übersichtsbäumchen und ermöglichen einen schnellen Zugriff.)

4. Überschrift des Hilfetextes:
Hier kann für einen Hilfetext, der bei Ein- und Ausgaben des Items (links unten) gezeigt werden soll, eine Überschrift eingegeben werden (ohne Anführungszeichen, weil diese Überschrift nicht durch ein Item ersetzt werden kann). Wenn eine Überschrift eingegeben wurde, öffnet sich ein weiteres Fenster für den Hilfetext:

5. Hilfetext:
Das ist ein mehrzeiliges Eingabefeld: mit <Strg Enter> kommt man in die nächste Zeile oder bricht die Zeile um.
Hier ist dann der Hilfetext einzutragen, und zwar ohne Anführungszeichen, weil der Text nicht durch ein Text-Item ersetzt werden kann. Für die Formatierung und für die Einfügung von Links sollte die Markdown-Konvention genutzt werden. Näheres und Links vgl. unten IV.4.

6. Bedingung für ein besonderes Ergebnis:
(Auch dies ist ein mehrzeiliges Eingabefeld: mit <Strg Enter> kommt man in die nächste Zeile oder bricht die Zeile um.) Das Eingabefeld für die Bedingung steht oberhalb des Feldes für eine Frage und gibt an, in welchen Fällen eine Frage nach dem Wert des Items gar nicht gestellt werden muss, weil sich der Wert des Items schon aus dem Wert anderer Items ergibt. Man kann also mit Hilfe dieser Bedingung vermeiden, dass eine Frage gestellt wird. Eine Vereinfachung bei der Bestimmung des Itemwertes ergibt sich allerdings nur dann, wenn die Items, welche die Bedingung bilden, bei Aufruf

---

[2] Wenn man mit Komma getrennt, in ein Eingabefeld die Funktion
"Topic_Soutp(<Text>,<Überschriftebene>,<Ausgabeform>)" einträgt, dann wird die Überschrift vor der Ausführung der Anweisung in dem Eingabefenster gemeldet.
- Text: einfach in Anführungszeichen setzen,
- Überschriftebene: ist eine Ganzzahl, von 0 bis... Doch meldet das Programm einen Fehler, wenn die Ebenen nacheinander aufgerufener Überschriften um mehr als eine Überschriftebene wachsen.
- Ausgabeformen sind: BILD, DRUCK oder DRUCKBILD, also eines dieser Worte ist in Kapitalien einzutragen.

Beispiel:
Das Eingabefeld "Frage:" sei gewählt worden:
Eintrag in dieses Eingabefeld:
Topic_Soutp("Einkommen", 2, DRUCKBILD), "Wie hoch ist das Einkommen der Mutter?"
Bemerkung: Die Überschrift „Einkommen“ wird nur ausgegeben, wenn das Programm die Anweisung dieses Feldes ausführt, also die Frage stellt. Auch wenn das Programm die Meldung "Einkommen der Mutter: 0" unterdrückt, um Platz zu sparen, bleibt die Überschrift stehen. Wäre das Eingabefeld für die Überschrift weiter oben verwendet worden, dann wäre die Überschrift wegen des Nullwerts beim Einkommen der Mutter gelöscht worden.

der Bedingung bereits wirklich bestimmt worden sind. Ist das nämlich nicht der Fall, dann bewirken die Bestimmungsfunktionen dieses Items, dass – nur für die Prüfung der Bedingung – der Itemwert – ggf. durch eine Frage - bestimmt wird. Will man dieses Ergebnis vermeiden, und nur dann die Bedingung gelten lassen, wenn sich deren Wert der auch ohne Fragen feststellen lässt, dann muss man die Zusatzbedingung stellen, dass das Item schon bestimmt worden ist. Das geschieht mit der Funktion Is_set(), welche zu jedem Item gehört. Die Bedingung lautet dann nicht einfach *Itemname,* sondern *Itemname.Is_set()* UND *Itemname.* Wie Bedingungen zu formulieren sind, ist i.Ü. unten bei den logischen Items IV.5. beschrieben. Wenn das Feld leer bleibt geht das Programm weiter zur h) Frage. Nur wenn eine Bedingung für ein besonderes Ergebnis eingegeben wurde, öffnet sich ein Feld für dieses besondere Ergebnis

7. Besonderes Ergebnis
Auch dies ist ein mehrzeiliges Eingabefeld: mit <Strg Enter> kommt man in die nächste Zeile oder bricht die Zeile um. Das Ergebnis muss dem Itemtyp entsprechen, kann aber ein Rechenwerk darstellen, wie weiter unten „beim Programmieren mit..“ dargestellt. Wenn das Ergebnis nicht nur ein Wahrheitswert ist, folgt die Frage

8. Ergebnistext
Hier ist die Meldung, der unter der bestimmten Bedingung erzielten Ergebnisses zu bestimmen. Das Programm schlägt bei Rechenausdrücken in den vier Grundrechenarten die Ausgabe eines Rechenwerks vor, welches das Rechenergebnis begründen kann.

Danach wird solange nach einer weiteren Bedingung gefragt, bis das Fragefeld am Ende leer gelassen wird. Dann geht das Programm weiter zu dem Eingabefeld.

9. Frage:
Hier kann der Itemwert durch Antwort auf eine Frage bestimmt werden, die aber nur gestellt wird, wenn keine der etwa vorher formulierten Bedingungen zutrifft. Wenn keine Frage formuliert wird und der Fragevorschlag des Programms gelöscht wird, geht das Programm gleich weiter zum Rechenausdruck. Wird ein Fragetext formuliert oder der vorgeschlagene Text übernommen, dann muss die Frage noch näher bestimmt werden. Die Fragetexte müssen in Anführungszeichen gesetzt werden, weil man dafür auch ein Text-Item verwenden kann.

10. Programmvorschlag:
Hier ist je nach Itemtyp ein bestimmter Wert einzutragen, den das Item annimmt, wenn man einfach den Programmvorschlag bestätigt. Dieser Vorschlag findet sich im erzeugten Programm dann im Eingabefeld der Frage als durch **Prg:** gekennzeichneter Programmvorschlag.

11. Nutzerentscheidung verlangen:
Hier muss entschieden werden, ob das Programm bei der Frage anhalten, oder vorerst mit dem Programmvorschlag weiterrechnen soll, sodass der Anwender nur nachträglich diese Eingabe ändern kann.

12. Die Antwort ist falsch, wenn:
Zur fachgerechten Programmierung gehört es, Fehleingaben des Letztanwenders möglich früh festzustellen und zu rügen, damit das Programm keine unverständlichen Ergebnisse liefert. Deshalb sollten bei jeder Frage fehlerhafte Eingabe blockiert werden. Dazu sind Bedingungen einzutragen, aus denen sich in falsches Ergebnis ergibt. So kann etwa das Erwerbseinkommen nicht größer sein als das Gesamteinkommen. Die Fehlerbedingung würde dann lauten:
Erwerbseinkommen>Gesamteinkommen. Ist eine solche Bedingung formuliert, dann öffnet sich die neue Frage.

13. Meldung bei Fehleingabe:
Hier ist einzutragen, was das Programm bei dem bezeichneten Fehler melden soll, z.B.: „Erwerbseinkommen kann nicht größer als das Gesamteinkommen sein!“. Die Anführungszeichen sind nötig, weil auch der Name eines Text-Items verwendet werden könnte.
Danach wird wieder nach einer Fehlerbedingung gefragt, bis dieses Fragefeld leer gelassen wird.

14. Rechenausdruck
Hier ist dann, wenn die Frage fehlt oder mit „ich weiß nicht“ beantwortet werden darf, ein logischer oder mathematischer Rechenausdruck je nach Itemtyp einzutragen. Bei Text-Items kann bedingte Textaddition verwendet werden (vgl. Programmieren mit Text-Items V. Danach folgt die Frage.

15. Überschrift nach Bestimmung des Item-Werts:
Weil vor der Bestimmung eines Item-Werts oft die Werte von vielen anderen Items bestimmt werden müssen, kann die weiter oben eingegebene Überschrift den Zusammenhang mit der eigentlichen Bestimmung des Items verloren haben. Dann kann es nützlich sein, vor der eigentlichen Bestimmung des Items eine besondere Überschrift einzufügen, welche hier einzutragen ist. Auch diese verschwindet, wenn keine Meldung folgt, vgl. oben b)

16. Text für ein berechnetes Ergebnis
Hier ist die Programmmeldung einzutragen, welche erfolgt, wenn das Ergebnis berechnet wurde. Ist das Feld leer, erfolgt keine Meldung. Anderenfalls muss noch die Ausgabebedingung formuliert werden:

17. Bedingung für die Ausgabe:
Hier kann man die Meldung des Ergebnisses z.B. davon abhängig machen, dass der Wert größer als null ist. Wegen der Möglichkeit, dass ein PC an der zehnten Stelle noch einen minimalen Wert statt der Null ausweist, ist es zweckmäßig, die Bedingung nicht als „Itemwert==0“ zu formulieren, sondern „Itemwert<DELTA“, weil DELTA ein winzige Zahl ist. Wenn auch negative Wert möglich sind und ausgeschlossen werden sollen, muss eine C-Funktion verwendet werden: „gleich(Itemwert,0)“.

18. Nach Wertbestimmung
Am Ende bietet das Programm noch die Möglichkeit an, die Bestimmung des Item-Werts noch eine Aktion anzuschließen, insbesondere die Bestimmung des Wertes eines anderen Items. Wenn hier der Name eines Items eingetragen wird, wird im Programmlauf nach der Wertbestimmung des ursprünglichen Items dessen Wert bestimmt. Es öffnet sich dann ein neues Feld, in welches weitere Items eingetragen werden können. Es können aber auch beliebige C/C++-Programme eingetragen werden, die dann im Anschluss an die Bestimmung des Items ausgeführt werden.

V. Programmieren mit Ja/Nein-Items (boolesche Items) und Aufzählungen (Menüeingaben)

Die einfachsten Zusammenhänge sind die ausschließlich logischen, nämlich die Ja/Nein-Zusammenhänge (englisch: boolean) bei denen nur die Werte **wahr** oder **falsch** möglich sind. Eine wesentliche Stärke von ExpertItems besteht nun darin, dass sich auch diese effizient programmieren lassen, sodass auch juristische Fragestellungen mit Nutzen umgesetzt werden können. Wie alle Items werden Ja/Nein-Items durch Eingabe eines Namens in das entsprechende Eingabefeld erzeugt

1. Name des Items

mit dem sie aufgerufen werden können. Umlaute, Leerstellen u.ä. sind als Itemnamen nicht zugelassen. Das Programm akzeptiert zwar auch eine solche Eingabe, wandelt diese aber in einen

zulässigen Itemnamen um, welcher als Überschrift für die weiteren Fragen zu dem erzeugten Item dient. Als nächstes wird mit einem Auswahlmenü der

2. Typ

festgelegt, voraussetzungsgemäß hier der Typ **Ja/Nein**. Anschließend kann der Experte in dem Feld

3. Überschrift vor der Bestimmung des Itemswerts

den vom Item etwa zu liefernden Textausgaben eine Überschrift zuordnen. Wenn das Item zur Bestimmung seines Werts auf noch nicht bestimmte andere Items zurückgreift und bewirkt, dass auch diese bestimmt werden, dann erscheint die Überschrift vor den Ausgabetexten dieser anderen Items.

Die Überschrift muss in Anführungszeichen („“) gesetzt werden, weil sie sonst als Name eines Items gelesen wird.

Das bedeutet, dass das **Topic** auch durch ein anderes Item von einem Text-Typ bestimmt werden kann, s.u. bei Items von einem Text-Typ.

Topic-Ebene: Eine Überschrift bewirkt auch eine Gliederungsebene in dem im Feld links oben im Programmlauf dargestellten Orientierungsbäumchen. Diese Gliederungsebene kann man verändern, indem man (durch Komma getrennt) vor den Text der Überschrift einfügt **Shift_level**(x), wobei x die Anzahl der Verschiebungen darstellt. „1“ verschiebt den Eintrag in eine untere Ebene, also nach recht, „-1“ in eine höhere Ebene, also nach links.

4. Überschrift des Hilfetexts

Jedem Item kann ein Hilfetext zugeordnet werden, welcher dem Letztanwender die vom Item etwa gestellte Frage oder eine Meldung des Items erläutert. Als erstes fragt das Programm nach der Überschrift des Hilfetexts. Nur wenn eine solche Überschrift eingegeben wurde öffnet sich ein weiteres (nun mehrzeiliges) Eingabefeld „Hilfetext:“, in welches dann der weitere Hilfetext für den Letztanwender eingetragen werden kann, entweder als HTML-Fragment oder mit "Markdown". Der Hilfetext bildet zusammen mit der Überschrift des Hilfetexts die vollständige Hilfe für das Item.

Eine neue Zeile wird erzeugt, wenn statt <Enter> (das führt in das nächste Eingabefeld), <Strg Enter> eingibt.

Markdown
Markdown ist eine einfache "markup" - Sprache, welche es gestattet, leicht lesbaren Text zu schreiben, welcher durch Kompilation in strukturell gültiges (X)HTML umgewandelt wird.

Überschriften
HTML Überschriften werden erzeugt, indem eine Anzahl "#" vor den Text gesetzt werden, z.B.
erste Ebene - Überschrift
vierte Ebene - Überschrift

Absätze

Ein Absatz besteht aus einer oder mehreren aufeinander folgende Textzeilen, die durch eine oder mehrere Leerzeilen getrennt sind. Normale Absätze sollten nicht mit Leerzeichen oder Tabs eingerückt werden. Dabei muss auch die Zeile umgebrochen werden, indem man am Zeilenende <Strg Enter> eingibt.

Dies ist ein Absatz. Er hat zwei Sätze.

Die ist ein anderer Absatz. Er hat auch
zwei Sätze.

Listen
Es gibt in Markdown zwei Listentypen, die geordnete Liste und die ungeordnete Liste, wie in HTML. Eine ungeordnete Liste wird erzeugt, indem man eine Anzahl Einrückungen und "Knollen" vor die Elemente der Liste setzt. "Knollen" können, Sternchen, Pluszeichen oder Minuszeichen sein. davor setzt.

* Ein Element in einer ungeordneten Liste
    + Ein Unterelement, vier Leerzeichen eingerückt
* Ein weiteres Element
* Und noch eines

In einer geordneten Liste brauchen die Nummern nicht zur Anzahl der Elemente zu passen.
z.B.
1. Ein Gegenstand einer geordneten Liste
    1. Ein Untergegenstand, viel Stellen eingerückt
2. Noch ein Gegenstand der nummerierten Liste
3. Noch ein Gegenstand der nummerierten Liste
4. Ein anderer Gegenstand

Hervorgehobener Text
Für Hervorhebungen (Kursiv) gibt es zwei Möglichkeiten, nämlich eingefügt in Sternchen oder Unterstrichen. Eingefügt in einzelne Sternchen oder Unterstrich bedeutet "kursiv".

kursiv oder _kursiv_

Eingefügt in zwei Sternchen oder Unterstriche bedeutet "fett".

fett oder __fett__

Eingefügt in drei oder mehr Sternchen oder Unterstriche bedeutet kursiv und fett zugleich.

Links
Auch Links können eingefügt werden, wobei der Link-Text in eckige Klammern und die Link-Adresse in runde Klammern eingeschlossen wird.

Da die Link-ID eines jeden Hilfetexts aus einer Überschrift herauskopiert werden kann, lassen sich Hilfetexte effizient verlinken, z.B. mit der Hilfe-ID "HF_TRootNode_Projekt_Name" lautet der Link zu dem entsprechenden Hilfefenster:

[vgl. Projektname](#HF_TRootNode_project_name)

Links auf Gesetzestexte

Links auf Gesetzestexte können erzeugt werden in dem URNs aus dem LEX Namensraum als Ziel verwendet werden.

Um zum Beispiel einen Link auf §19 III VersAusglG zu erzeugen geben Sie folgendes ein:
[§19 III VersAusglG](urn:lex:de:gesetz:versausglg19;3)

Die Absatznummer ist optional und kann weggelassen werden:
[§21 BAföG](urn:lex:de:gesetz:bafoeg21)

5. Bestimmungsfunktionen:

Für die Bestimmung des Itemwerts stehen die Felder **Bedingung für ein besonderes Ergebnis/Besonderes Ergebnis, Frage** und **Rechenausdruck** zur Verfügung:

Hat ein Item mehr als eine Bestimmungsfunktion, dann können sie alle mit UND und ODER zu einer einzigen Bestimmungsfunktion zusammengefasst werden. Das geht jedoch dann nicht, wenn eine der Bestimmungsfunktionen die **Frage** ist. Dann muss deren Verhältnis den anderen Bestimmungsfunktionen festgelegt werden. Zwischen **Frage** und einem Rechenausdruck (**Rechenausdruck** oder **Besonderes Ergebnis**) sind zwei Beziehungen möglich: entweder hängt die Rechenfunktion davon ab, dass die Frage mit „ich weiß nicht“ beantwortet wird oder die Rechenfunktion tritt an die Stelle der Frage, so dass diese gar nicht erst gestellt wird. In diesem Fall gibt es wieder zwei Möglichkeiten: entweder gibt es überhaupt keine Frage, oder sie entfällt, wenn eine bestimmte Bedingung erfüllt ist. Den ersten und zweiten Fall deckt die **Rechenausdruck** ab, den dritten die **Bedingung für ein besonderes Ergebnis** mit der **Besonderes Ergebnis**.

Die Berechnungsfunktionen: **Rechenausdruck** und **Besonderes Ergebnis** sind also beide von einer typischen Bedingung abhängig, die sie jedoch unterscheiden: die **Rechenausdruck** hängt von der **Frage** und die **Besonderes Ergebnis** von der **Bedingung für ein besonderes Ergebnis** ab. Sonst sind sie völlig gleich.

Diese notwendigen Leistungen bei Zusammentreffen von Frage und Rechenausdruck werden ergänzt durch eine nützliche Leistung:

Die **Bedingung für ein besonderes Ergebnis** kann nämlich wiederholt werden. Anstelle eines Komplexausdrucks mit *ODER* kann an eine **Bedingung für ein besonderes Ergebnis** eine oder mehrere weitere angeschlossen werden, was die Übersichtlichkeit verbessert. Die Wiederholung von Bedingungen kann aber auch verwendet werden, wenn überhaupt keine Frage gestellt wird: Dann dient sie der Aufteilung eines einzigen komplexen Bedingungszusammenhangs in Teilbedingungen der sog. „disjunktiven Normalform“ der booleschen Logik.

Hier soll mit der

a) Frage begonnen werden, weil alle Bestimmungsfunktionen am Ende zu Fragen führen müssen. Die Frage wird erzeugt, indem in das Feld

aa) Frage ein Text eingegeben wird. Dieser Text muss in Anführungszeichen („“) stehen, sonst wird er als Name eines Items gelesen (Bezugnahme auf ein Text-Item ist nämlich ebenfalls möglich ist, s.o. 3.). Das Programm schlägt als Fragetext den Itemnamen vor. Diesen wird man bei der Programmierung zur Vereinfachung vorläufig verwenden, in der Endfassung aber durch einen besser geeigneten Text ersetzen.

Für die Eingabe des Itemwerts soll das geplante Programm in der Regel einen Vorschlag liefern, den

bb) Programmvorschlag (ja/nein/unbestimmt). Dieser kann <ja>, <nein> oder <unbestimmt> lauten. Wenn der Endbenutzer im erzeugten Programm <unbestimmt> wählt, so wird der Item-Wert durch die **Rechenausdruck** (s.u.) bestimmt. Wird <unbestimmt> als **Programmvorschlag** gewählt so muss daher ein **Rechenausdruck** (s.u.) eingegeben werden.

Wenn ein Vorschlag (meist wird man dafür den wahrscheinlichsten Fall wählen) vorhanden ist, kann das Programm ohne Eingreifen des Anwenders vorläufig weiterrechnen. Das ist aber nicht immer sinnvoll. Durch Bejahung der folgenden Frage:

cc) Nutzerentscheidung verlangen kann dieses automatische Weiterrechnen verhindert werden. Das Programm liefert dem Anwender dann die Meldung „Eingabe ändern oder bestätigen“ und wartet auf eine Eingabe.

Wenn eine **Frage** fehlt oder der **Programmvorschlag** durch eine Leereingabe mit *unbekannt* beantwortet wurde, dann muss der Wert des Items berechnet werden. Dazu dient der

b) Rechenausdruck.

Wird voraussetzungsgemäß nur mit booleschen Items gearbeitet, stehen auch nur solche für die Berechnung des Ja/Nein-Werts zur Verfügung. Die Berechnung erfolgt nach der booleschen Algebra, wobei bestimmte umgangssprachliche Ausdrücke akzeptiert werden: Die **Rechenausdruck** eines Items *„Erbe“* könnte daher z.B. lauten

„Vertragserbe ODER testamentarischerErbe ODER gesetzlicherErbe UND NICHT enterbt“,

wenn *„testamentarischerErbe“*, *„gesetzlicherErbe“* und *„enterbt“* Ja/Nein-Items sind. Statt *ODER*, *UND* und *NICHT* können auch *OR*, *AND* und *NOT* oder „||“, „&&“ und „!“ verwendet werden. Da *UND* gegenüber *ODER* vorrangig ist, müssen mit *ODER* verknüpfte Items ggf. in Klammern gesetzt werden, also z.B.

(Vertragserbe ODER testamentarischerErbe ODER gesetzlicherErbe UND NICHT enterbt) UND NICHT ausgeschlagen,

wenn auch *ausgeschlagen* ein Ja/Nein-Item ist.

Da *ODER* und *OR* ebenso wie || der booleschen Algebra folgen, bedeuten sie nicht *entweder … oder …*, sondern sie schließen auch die Möglichkeit ein, dass beide Teilaussagen wahr sind.

Als **Rechenausdruck** kann auch jeder beliebige C- oder C++-Programm eingetragen werden, welches das Ergebnis mit *return xxx;* zurückliefert.

Wird eine **Rechenausdruck** eingetragen, dann muss das erzeugte Programm bei einer etwaigen **Frage** die Möglichkeit bieten, *unbestimmt* einzugeben, also *ich weiß nicht* zu melden, was dann dazu führt, dass der Itemwert berechnet werden muss und dabei ggf. auch die in der **Rechenausdruck** aufgerufenen Items bestimmt werden müssen, entweder durch Fragen oder wieder durch andere Items. Dadurch können stufenweise weitere Informationen abgefragt werden.

Im Gegensatz zu maskenbasierten Systemen werden die Fragen durch die vorherigen Eingaben gesteuert.

Der Rechenausdruck kann auch dazu verwendet werden, die Frage selbst mit Bedingungen hinsichtlich des Eingabe-Vorschlags zu versehen. Wenn z.B. in der Unterhaltsvariante I von WinFam in dem Feld „umgeb1(var.drittelohnebonus)[0]“ ein ‚N‘ also „nein“ eingetragen ist kann damit der Eingabevorschlag gesteuert werden, indem in das Feld für den Rechenausdruck

```
Bool_input("Drittelberechnung ohne Bonus?",umgeb1(var. drittelohnebonus)[0] == 'N', false, false)
```

eingetragen wird. Das erst „false“ bedeutet dass bei dem Eintrag „N“ in der Variante „nein“ also false, vorgeschlagen wird, das zweite „false“ bewirkt, das Programm mit der Frage nicht anhalten soll, sondern mit der Standardvorgabe weiterrechnen darf. Wird stattdessen „true“ eingetragen, dann hält das Programm bei dieser Frage an. Man kann aber ohne Rückgriff auf die C-Programmierung auch abhängig von der Bedingung zwei verschiedene Frage-Items verwenden.

Wenn durch andere Items der Wert des Items bereits bekannt ist, kann mit

c) Bedingung für ein besonderes Ergebnis

Die Bestimmung des Items kann durch eine vorhandene **Frage** vermieden werden. Jede Bedingung ist eine Ja/Nein-Funktion und besteht wieder aus Ja/Nein Items, wenn nur Items dieses Typs zur Verfügung stehen. Der Aufruf der Items zur Bestimmung der Bedingung bewirkt, dass der Wert dieses Items bestimmt wird.

Will man die **Frage** nur dann vermeiden, wenn die Items der **Bedingung für ein besonderes Ergebnis** bereits vorher bestimmt worden waren, dann muss vorher geprüft werden, ob das Item auch schon bestimmt wurde. Diese Information findet man in

Itemname.is_Set(), wenn *Itemname* der Name des Items ist. Die Bedingung lautet dann *Itemname.is_Set() UND Itemname*.

Wenn man wissen will, ob das **andere Item direkt eingegeben oder anders bestimmt wurd**e, kann man die Bedingung ***Itemname.Value_from_input()*** verwenden.

Wird eine **Bedingung für ein besonderes Ergebnis** eingegeben, dann öffnet sich ein weiteres Eingabefeld und es muss ein

d) Besonderes Ergebnis

eingegeben werden. Bei einem Ja/Nein-Item kann das nur ein Wahrheitswert sein, also „true“ oder „false“. Solange das **Besondere Ergebnis** nicht eingegeben ist, ist die weitere Programmierung blockiert. Sobald aber als **Besonderes Ergebnis** ein Wert eingegeben wurde, öffnet sich ein neues Eingabefeld für eine weitere **Bedingung für ein besonderes Ergebnis.** Das wiederholt sich, bis das Eingabefeld für **Bedingung für ein besonderes Ergebnis** leer gelassen wird.

Damit kann eine Kette alternativer Anspruchsvoraussetzungen programmiert werden: Jede Bedingung stellt ein Item dar und dessen Bestimmungsfunktion ist jeweils eine Ja/Nein-Frage.

In den meisten Fällen muss das Ergebnis der Bestimmung des Itemwerts auch irgendwie gemeldet und in das Ergebnisprotokoll oder den zu erzeugenden Text eingetragen werden.

Davor kann aber noch eine Überschrift eingetragen werden, nach welcher das Programm mit

6. Überschrift nach der Bestimmung des Itemwerts

fragt. Es kann nämlich sein, dass die **Überschrift vor der Bestimmung des Itemswerts,** das die Bestimmung des Items eingeleitet hatte, durch die Ausgabe anderer Itemwerte, die vorher bestimmt werden mussten, zu weit von dem durch die Überschrift gekennzeichneten Item entfernt wurde. Die **Überschrift nach der Bestimmung des Itemwerts** steht dann unmittelbar vor den Ergebnisausgaben, die durch die Bestimmung des Items selbst ausgelöst wurden. Eine hier eingetragene Überschrift wird deshalb erst ausgegeben, wenn etwaige Meldungen aus anderen Items, welche zur Bestimmung des aktuellen Items aufgerufen wurden, bereits ausgegeben wurden.

Wie die **Überschrift vor der Bestimmung des Itemswerts** muss der Text auch dieses Topic mit Anführungszeichen („“) eingegeben werden, weil er sonst als Itemname gelesen wird.

Die Überschrift bewirkt auch eine Gliederungsebene in dem im Feld links oben im Programmlauf dargestellten Orientierungsbäumchen. Diese Gliederungsebene kann man verändern, indem man (durch Komma getrennt) vor den Text der Überschrift einfügt **Shift_level**(x), wobei x die Anzahl der Verschiebungen darstellt. „1“ verschiebt den Eintrag in eine untere Ebene, also nach recht, „-1“ in eine höhere Ebene, also nach links.

7. Standard-Ergebnistext

gemeldet werden**.** Da der Wert eines booleschen Items nur *true* und *false* sein kann, kommen zwei Meldungen in Betracht, nämlich

a) Ausgabe für „ja“,

für den das Programm *<itemname> = true* vorschlägt, und

b) Ausgabe für „nein“,

für den das Programm *<itemname> = false* vorschlägt.

Diese Vorschläge müssen durch Aussagesätze ersetzt werden, welche die Bejahung oder die Verneinung des Items beinhalten, z.B. „Die Voraussetzungen für das Bestehen eines Kaufpreisanspruchs sind erfüllt.“ oder „Die Voraussetzungen für das Bestehen eines Kaufpreisanspruchs sind nicht erfüllt.“. Oft interessiert nur die Bejahung wenn etwa eine lange Liste alternativer Voraussetzungen zu prüfen ist. Dann muss der programmierende Experte den verneinenden Text ersatzlos löschen. Oder umgekehrt ist der bejahende Text zu löschen, wenn bei sonst bejahten Voraussetzungen nur eine letzte Bedingung das Ergebnis hindert. Viele weitere Möglichkeiten ergeben sich daraus, dass man den Ausgabetext durch ein ihn enthaltendes Text-Item bezeichnet. Das ermöglicht es, den Ausgabetext weiter zu gestalten.

Die Ausgabe eines Ergebnistextes kann auch von einer Bedingung abhängig gemacht werden, der

c) Bedingung für die Ausgabe.

Hier kann ebenso wie bei der **Bedingung für ein besonderes Ergebnis** eine beliebige Bedingung eingetragen werden, z.B. davon, dass mehrere andere Items bejaht worden waren.

Der **Standard-Ergebnistext** ist nur ein Unterfall einer Aktion

8. Eingreifen in den automatischen Programmablauf - ergänzendes Programmieren

a) Eingabefolge

Wenn der auf Grund der logischen Zusammenhänge automatisch erzeugte Programmablauf den Anwendererwartungen nicht entspricht, dann muss die Abfolge der Fragen korrigiert werden. An sich wird ein Item erst dann bestimmt, also eine etwaige Frage erst dann gestellt, wenn das Ergebnis benötigt wird. Soll die Frage aber vorher gestellt werden oder überhaupt der Wert eines Item eher bestimmt werden, als der dem automatisch erzeugten Programm entspricht, dann kann dies erzwingen werden, in dem das Item in dem dann folgenden Item vorweg aufgerufen wird. Das geschieht so, dass in dem ersten Eingabefeld mit einem Eintrag, z.B. **Bedingung für ein besonderes Ergebnis** vor den Eintrag, durch Komma getrennt, der Name des vorher zu bestimmenden Items geschrieben wird. Der Name muss mit den Klammer, z.B. „Antrag_zugegangen()“ versehen sein, um darauf hinzuweisen, dass hier ein C++-Befehl auszuführen ist. Sollen mehrere Items vorher bestimmt werden, können auch mehrere Items mit Klammern, durch Komma getrennt, dort eingetragen

werden. Wenn bereits eine Überschrift eingegeben wurde, kann der Aufruf des anderen Items auch im Eingabefeld für die Überschrift erfolgen.

Wenn allerdings ein Item im Anschluss an das Item bestimmt werden soll, ist ein besonderes Eingabefeld erforderlich:

b) Nach Wertbestimmung:

Wenn in dieses Feld der Name eines Items eingetragen wird, so wird dasselbe immer Anschluss an das andere Item bestimmt. Hier müssen allerdings nicht nur die Klammern an den Item-Namen angefügt werden. Es muss auch ein „;“ angefügt werden als Hinweis darauf, dass es sich um einen C++-Befehl handelt, z.B. „Antrag_zugegangen();“.

c) Textausgaben:

In das Eingabefeld „Nach Wertbestimmung“ können Textausgaben, aber auch beliebige andere Aktionen im Anschluss an die Bestimmung des Itemwerts und einen etwaigen **Standard-Ergebnistext** veranlasst werden.

Hier – wie auch sonst überall – stehen die Möglichkeiten der C++-Programmierung zur Verfügung. Insbesondere die Bedingung, z.B.

„if (Antrag_zugegangen) String_output(„Der Antrag ist zugegangen.“);

Jedes C- oder C++-Statement ist hier zulässig. Es können aber auch einfach Items aufgerufen (s.o.) und die Bestimmung ihres Werts erzwungen werden, wenn man nicht warten will, bis Näheres zu den Texten bei den Items von Texttypen.

9. Aufzählungen (Menüeingabe)

Statt einer Auswahl nur zwischen „ja“ und „nein“ können auch weitere Möglichkeiten zur Auswahl stehen.

Man könnte dann nacheinander alle Möglichkeiten mit ja und nein abfragen. Effektiver ist es aber eine **Menüabfrage**, welche sofort eine aus mehreren Möglichkeiten zu wählen gestattet. Die **Aufzählung** gehört zu den „sonstigen“. Soll das Item eine Aufzählung liefern, dann ist zuerst „**sonst**“ zu wählen. Sodann erscheint eine Liste, in welcher am Ende „**Aufzählung**“ erscheint.

Die möglichen Werte, welche im Eingabemenü erscheinen sollen, werden vom Programm nacheinander abgefragt:

Name des 1. Wertes

Name des 2. Wertes

….usw

Die Frage wird wiederholt, bis keine Eingabe mehr erfolgt. Alle eingegebenen Texte, z.B. „Gesetzliche Rentenversicherung“, erscheinen für den Letztanwender dann untereinander im Eingabemenü. Bei der Weiterverwendung des Ergebnisses ist nicht der eingegebene Text zu verwenden, sondern eine frei wählbares Symbol, für das das Programm den eingegebenen Text in Kapitalien und ohne Leerzeichen (stattdessen mit Unterstrichen) vorschlägt.

VI. Programmieren mit Text-Items

Text-Items sind Items der

1. Typen: Text.

Textitems können **einzeilig** und **mehrzeilig** sein.

Der Unterschied zwischen beiden zeigt sich nur bei der

2. Frage:

Wenn der Wert eines Text-Items (auch) durch eine **Frage** bestimmt werden kann, also ein Fragetext eingetragen wird, dann erscheint vier Zeilen weiter unten die Frage

mehrzeilig?

Wird diese Frage bejaht dann befindet sich das Eingabefeld für den Letztanwender **unterhalb des Fragefelds.** Sonst befindet sich das Eingabefeld **rechts neben dem Fragefeld**, wodurch die Möglichkeit, längere Texte einzugeben, stark beeinträchtigt wird. Bei **mehrzeilig** können aber beliebig viele Zeilen eingegeben werden. Einen Zeilenumbruch ist mit <strg return> zu bewirken. Die Eingabe des Textes durch den Endanwender erfolgt ohne Anführungszeichen. Item-Namen sind als Antwort auf eine **Frage** deshalb nicht zugelassen.

Mehrzeilige Eingaben verwendet auch das Programmierprogramm **ExpertItems**. Auch der dieses Programm verwendende Experte muss <strg return> verwenden, wenn er eine neue Zeile erzeugen oder eine Zeile umbrechen will. Als Antwort auf die Fragen, welche ExpertItems an den programmierenden Experten stellt, sind jedoch auch Item-Namen zugelassen. Folglich müssen Texte durch Anführungszeichen „“ als solche gekennzeichnet werden.

Bei der

3. Bedingung für ein besonderes Ergebnis, der **Bedingung für die Ausgabe** oder **Die Antwort ist falsch, wenn ...**

kommen die **Vergleichsoperatoren** für Texte in Betracht**.** Mit **==** (gleich) und **!=** (ungleich) können Texte verglichen werden. z.B. bedeutet

Irgendein_Item!=„“,

dass das Text-Item mit dem Namen *Irgendein_Item* nicht leer („“) ist.

Name==“Anton“

bedeutet, dass das Item mit dem Namen *Name* den Wert *Anton* hat. Für

4. Rechenausdruck und **Besonderes Ergebnis**

sind von größter Bedeutung die

Textaddition**:** z.B.

Kind + (string) „ ist ein Kind von „+ Vater + (string)“.“

Man kann auch schreiben

Kind + string(„ ist ein Kind von „)+ Vater + string(“.“)

Wenn *Kind* ein Item des Typs **Text** ist mit dem Inhalt *Anton*, und *Vater* ein Item des Typs **Text** ist mit dem Inhalt *Martin*, entsteht durch die Addition der Text

Anton ist das Kind von Martin.

Wenn in eine solche Textaddition einfache Zeichenketten, z. B. *„ ist ein Kind von „* oder der Punkt *„.“* einbezogen werden, müssen durch Voranstellen des Umwandlungsoperators **(string)** in einen Text umgewandelt werden (wie oben geschehen), welche dann addiert werden kann: z.B. *(string) „Unterhalt von „+Name_des_Berechtigten+ „.“*.

Es genügt allerdings, dass die Umwandlung einmal am Anfang erfolgt.

Bei der Textaddition sind die zu addierenden Textteile normalerweise durch Leerstellen voneinander getrennt. Diese müssen normalerweise in die Zeichenketten bei Beginn oder/und Ende eingetragen werden, z.B. „ xxx „.

Wichtig ist die Verwendung der erzeugten Texte in Ausgabefunktionen.

5. Nach Wertbestimmung,

z.B. können mit

String_output(<irgendein Text>);

Currency_output(<irgendein Text>, <irgendein Item des Typs **Geldwert**>);

Double_output(<irgendein Text>, <irgendein Item des Typs **floating point**>);

Percentage_output(<irgendein Text >, <irgendein Item des Typs **percentage**>);

einzelne Ergebnisausgaben angeordnet werden.

Table_output(<irgendeine Tabellenzeile als String oder C-string>**)**ermöglicht die Ausgabe von **Tabellen**, weil die Buchstabenabstände beim Ausdruck unverändert bleiben.

z.B. **Currency_output**(*(string)"Unterhaltsanspruch von „+Kind+" gegen „+Vater+ „:", Anspruch*);

Wenn *Kind* und *Vater* Items vom Typ **Text** sind und Anspruch ein Item vom Typ **Geldwert**. Wenn das Item *Kind* den Wert ‚Miriam' hat, *Vater* den Wert ‚Martin' und ***Anspruch*** **den Wert ‚400' , dann liefert dieser Ausdruck den Text: *Unterhaltsanspruch von Miriam gegen Martin: 400 Euro*.**

Bei der Textaddition können neben Items auch einfache Zahlen oder Variable von Bedeutung sein, die sich nicht ohne weiteres für die Addition in Texte umwandeln lassen. Erscheint dann die Fehlermeldung „Typumwandlung … kann nicht konvertiert werden", dann ist eine Konvertierungsfunktion zu verwenden, deren Name ihre Funktion andeutet:

Int2string(<Ganzzahl>) wandelt eine Ganzzahl in einen Text um,

Cu2string(<Geldbetrag>) wandelt einen Geldbetrag in einen Text um**,**

Table_cu2string(<Geldbetrag>) tut dasselbe für Tabellenausgaben,

D2string(Kommazahl) wandelt eine Kommazahl in einen Text um**,**

D2string2(Kommazahl) wandelt eine Kommazahl zweistellig in einen Text um,

D2string5(Kommazahl) wandelt eine Kommazahl fünfstellig in einen Text um**,**

Prc2string(Kommazahl) wandelt die Kommazahl in einen Prozentbetragstext um, ebenso

Prc2string(Kommazahl, Ganzzahl), wobei hier nur die angeschlossene Ganzzahl die Anzahl der Nachkommastellen angibt,

die mit Stringaddition addiert werden können,

(die „2" in den Bezeichnungen steht dabei für das englische „to").

Wenn man in C programmieren will, muss man dafür den C-String des Items verwenden. Dieser lautet: „Itemname.c_str()"

6. Additionsketten:

Vielfach kann man einen Betrag erläutern, indem man in den erklärenden Text einen additiven Ausdruck einfügt, der die bereits oben berechneten additiven Beträge enthält. Diese werden aber nicht benötigt, wenn sie nicht vorhanden sind bzw. mit „0" valutieren.

Hier kann **Add_string**(double x) verwendet werden. Hier wird der Betrag (mit „+" oder „-„) nur dann an den Textstring angefügt, wenn er einen von „0" verschiedenen Wert hat.

Z.B. „Das Einkommen von "+Name_der_Partei+ „ beträgt: "+D2string(Brutto)+Add_string(-Aufwendungen)+Add_string(-Kindesunterhalt)+"=", was einen entsprechenden Text liefert, z.B.

„Einkommen von Vater beträgt 3000-600-300="

Wenn Aufwendungen und Kindesunterhalt von 0 verschieden sind. Sonst fallen diese Glieder einfach weg.

7. Zeilenumbruch, Leerzeile

Wenn der ausgegebene Text umgebrochen werden soll, kann man das dadurch erreichen, dass man vor einen Eintrag, z.B. des Textergebnisses, mit Komma getrennt den Befehl „New_line()" setzt.

Man kann auf diese Weise auch Leerzeilen einfügen. Regelmäßig will man aber vermeiden, dass durch Zufallskombinationen sich Leerzeilen häufen. Dagegen hilft:

Empty_line_if_required()

Dieser Befehl kann, durch Komma getrennt, vor jeden Befehl in einer Eingabezeile gesetzt werden und sorgt dafür, dass dort eine Leerzeile eingefügt wird, sofern eine solche noch nicht vorhanden ist.

8. Reine Bildschirmausgabe

Wenn man Eingabefehler des Letztanwenders durch Hinweise vermeiden will, braucht man Textausgaben, welche nur auf dem Bildschirm erscheinen. Hierfür ist

Show_string() vorgesehen. In die Klammer kann ein Text-Item oder ein Text eingefügt werden. Z.B. Show_string(Kind[i].name) oder Show_string(„Alexander"). Weitere Bildschirmausgaben im Anhang: Fehlerbehandlung.

9. Reine Druckausgabe

Man kann auch den Bildschirm entlasten, indem man einzelne Ausgabe auf den Ausdruck beschränkt. Hierzu ist dem Ausgabebefehl die Befehl Next_output_print_only() mit Komma getrennt, voranzustellen, z.B.

Ergebnistext Eingaben: Next_output_print_only(), "Einkommen des Mannes"

10. Einfügen einer Leerzeile

VII. Programmierung mit Items eines quantitativen Typs

In vielen Fällen haben die Items nicht nur den Wert *true* oder *false*, also *1* oder *0*, sondern noch viele Werte darüber und darunter oder auch viele Werte dazwischen. Solche einfach quantitativen Typen sind der Typ

-Ganzzahl mit (positiven oder negativen) ganzzahligen Werten, z.B. *7* oder *-3*. Die positiven eignen sich zum zählen,

- Kommazahl mit (positiven oder negativen) Dezimalbrüchen oder sonstigen rationalen Zahlen, z.B. *0,7* oder *3/7*,

- Geldbetrag für Geldbeträge, also zweistellige Dezimalbrüche mit Größenbezeichnung „Euro".

-Prozentsatz für Dezimalbrüche, die in Prozentwerten ausgegeben werden.

Items mit einfach quantitativem Typ eröffnen viele neue Bestimmungsmöglichkeiten. Aus dem Menü der

1. Typen

muss also **Ganzzahl, Geldbetrag oder Prozentsatz ausgewählt** werden.

2. Bedingung für ein besonderes Ergebnis, Bedingung für die Ausgabe

kommen nun nicht nur die Werte der booleschen Items in Betracht, sondern auch ein Größenvergleich einem Items mit einem festen Betrag oder zwischen zwei Items, z.B.

Gesamteinkommen > 0

oder

Erwerbseinkommen < Gesamteinkommen

wenn *Erwerbseinkommen* und *Gesamteinkommen* Itemnamen sind.

Die Größenvergleiche liefern Ja/Nein-Werte, die ebenso wie die booleschen Items zu Ja/Nein-Funktionen verknüpft werden können, z.B.

Jahreseinkommen>8130 UND Jahreseinkommen< 13470

für den unteren Proportionalbereich der Einkommensteuer nach § 32a Abs.1 S.1 EStG oder

Erwerbseinkommen > Gesamteinkommen

3. Frage

Wenn der Letztanwender nach einer Quantität gefragt wird, darf er auch mit einem Rechenausdruck mit den vier Grundrechenarten antworten, also z.B. „25769/12“. Wenn der programmierende Experte wünscht, dass dieser Rechenausdruck auch später zur Verfügung steht, muss er dafür sorgen, dass er bei dem Item gespeichert, also Teil des Items wird. Dazu dient die Funktion **Next_input_store_calculations().** Sie bewirkt, dass das Rechenwerk bei dem Item bei **Item.Get_calculation()** zu finden ist, d.h. der Rechenausdruck mit dem Rechenergebnis, verknüpft durch das Gleichheitszeichen. Daneben findet sich auch **Item.Get_embraced_calculation(),** bei dem der Ausdruck in Klammern gesetzt ist. **Item.Get_calculation() und Item.Get_embraced_calculation()** sind Texte. Ihre Verwendung ist im Kapitel **VI. Programmieren mit Text-Items** erläutert.

Beispiel:
Name des Items: Eigeneinkommen
Frage: Next_input_store_calculations(), „Einkommen des Berechtigten“.
Ergebnistext Eingaben: „Einkommen des Berechtigten:“+ Eigeneinkommen.Get_calculation()
Wird in dem fertigen Programm vom Letztanwender „1500/12“, eingegeben, dann meldet das Programm: „Einkommen des Berechtigten: 1500/12 = 125“

Anders als bei den Ja/Nein-Items muss bei quantitativen Items immer auch eine

4. Die Antwort ist falsch, wenn ...(Antwort ist falsch, wenn …)

Bedingung eingegeben werden, unter welcher die Eingabe als falsch zu bewerten ist. Für Eingaben sollte immer eine Fehlerkontrolle erfolgen.

Die Antwort ist falsch, wenn ... bei der Eingabe des Erwerbseinkommens, könnte lauten: “Erwerbseinkommen > Gesamteinkommen“, weil das Erwerbseinkommen nie das Gesamteinkommen überschreiten kann.

5. Bei **Rechenausdruck** und **Besonderes Ergebnis**

kommen nun alle mathematischen Operationen in Betracht. Diese können durch alle C- oder C++-Funktionen realisiert werden. Praktisch bedeutsamer ist (vor allem im Rechtsbereich) die Bestimmung des Itemwerts durch die vier Grundrechenarten, z.B:

*(Einkommen_des_Mannes – Einkommen_der_Frau)*3/7*

für die Unterhaltsberechnung nach der Differenzmethode der Düsseldorfer Tabelle. Besonders beim

6. Standard-Ergebnistext

kann das Programm hier viel zur Transparenz des Rechenwegs beitragen: Wenn nämlich als **Rechenausdruck** oder als **Besonderes Ergebnis** ein Rechenausdruck nach den Grundrechenarten eingetragen wurde, dann schlägt das Programm einen **Standard-Ergebnistext** vor, welcher in dem daraus erzeugten Anwenderprogramm einen Rechenausdruck mit den fallbezogenen Daten erzeugt.

Aus

*(Einkommen_des_Mannes – Einkommen_der_Frau)*3/7*

würde dann im Rechenlauf z.B.

*Gattenunterhalt: (4000 – 1000)*3/7*

herauskommen.

7. Runden

Das Runden ist eine Funktion, die oft gebraucht wird. Expertitems bietet Hierfür **drei** Funktionen an:

Round(x,y) rundet kaufmännisch, d.h. heißt, die Zahl x wird auf y Stellen nach dem Komma gerundet. Ist die x+1. Stelle eine 5, so wird aufgerundet. Ist x negativ, dann wird vor dem Runden das Vorzeichen weggedacht. Dadurch wird bei Gegenbuchungen in die gleiche Richtung gerundet.

Round_up(x,y) rundet die Zahl x auf y Stellen auf. Negative Zahlen werden dann ich Richtung Null gerundet.

Round_down(x,y) rundet die Zahl x auf y Stellen ab. Negative Zahlen werden dann in Richtung weg von der Null gerundet.

VIII. Programmieren mit Items des Typs Datum

Der Datentyp **Datum** kann nach den Konventionen von C++ benutzt werden, d.h. der ExpertItems-Typ **Datum** und der gleichnamige C++-Typ sind identisch. Wird beim Programmieren ein konkretes Kalenderdatum verwendet, **müssen Anführungszeichen** verwendet werden (z.B. „1.8.1937“).

Die Elemente dieses Typs sind die Ganzzahlen (Ganzzahl) *d.*year, *d.*month und *d.*day, wenn d ein Item des Typs **Datum** ist.

1. Bedingung für ein besonderes Ergebnis, der **Bedingung für die Ausgabe** oder der **Die Antwort ist falsch, wenn ...** können die **Vergleichsoperationen** des Item-Typs **Datum** verwenden:

Z.B. bedeutet

Date1>Date2

dass das Datum des Items Date1 später ist als das Datum von Date2.

Date1>**date**(*„16.4.2011“*)

bedeutet, dass das Datum des Items Date1 später ist als das Datum *16.4.2011*.

Entsprechend können auch == (gleich), != (ungleich), < (früher), >=(später oder gleich) etc. verwendet werden.

date() wandelt eine Zeichenkette in ein **Datum** um, wie im zweiten Beispiel dargestellt.

2. Frage und unvollständige Eingaben

Gerade bei Kalenderdaten kann der Letztanwender sich die Arbeit durch Eingabe eines unvollständigen Datums erleichtern, Wenn es letztlich nur auf die Jahre ankommt, beschränkt er sich darauf, nur die beiden letzten Ziffern der Jahreszahl einzugeben. ExpertItems unterstützt ein solches Verfahren, indem es unvollständige Datums-Eingaben automatisch ergänzt und an die Stelle der fehlenden Zahl bei der Eingabe eine „1“ setzt.

Leereingaben sind nur erlaubt, wenn ein Rechenausdruck vorhanden ist.

Der programmierende Experte kann die automatische Ergänzung verhindern, indem er vor den Fragetext, durch Komma getrennt **Complete_date_only** schreibt, z.B. *Complete_date_only, „Datum der Geburt“*.

3. Die Felder **Rechenausdruck** und **Besonderes Ergebnis** können

a) Daten umwandeln, z.B. wird ein Datum ***d*** durch

***d*.Plus_months**(<Zahl der Monate>)

in ein um die entsprechende Anzahl von Monaten späteres Datum umgewandelt.

***d*.Minus_months**(<Zahl der Monate>)

in ein um die entsprechende Anzahl von Monaten früheres Datum umgewandelt.

***d*.Plus_days**(<Zahl der Tage>)

in ein um die entsprechende Anzahl von Tagen späteres Datum umgewandelt.

***d*.Minus_days**(<Zahl der Tage>)

in ein um die entsprechende Anzahl von Tagen früheres Datum umgewandelt.

***d*.Last_day_in_month()**

in das Datum des letzten Tages im Monat umgewandelt.

***d*.First_day_in_month()**

First_day_in_month(*d*)

in das Datum des ersten Tages im Monat umgewandelt.

d.*Last_day_previous_month*()

in das Datum des letzten Tages im vorherigen Monat umgewandelt.

***d.*First_day_next_month()**

in das Datum des ersten Tages des Folgemonats umgewandelt.

Kommentiert [WG1]:

b) Die Differenz von zwei Daten in Monaten (Typ Ganzzahl) ermitteln:

Kommentiert [WG2]:

d.*Difference_months(''15.1.90''), auch für Berechnung des Alters zu verwenden***
d.*Difference_months(''15.1.90'')/12***

*d.***Get_julian()** liefert die Anzahl der Tage ab Beginn der in der Wissenschaft üblichen sog julianischen Zeitrechnung, nämlich 1.1.4713 v. Chr..

*d.***Get_julian() –** *d_1.***Get_julian()** die Differenz zwischen den Daten *d* und *d_1* in Tagen, die durch Division durch 365,24 in Jahre umgewandelt werden kann (es ist dann zweckmäßig zu runden: round((d.Get_julian()-d_1.Get_julian())/365.24,-3).

d.**Number_of_workdays**(first, last, withSaturday) liefert die Anzahl der Werktage zwischen zwei Daten.

c) **Today()**

Wird ein Datum gefragt, kann mit **Today()** das heutige Datum als Eingabevorschlag eingetragen werden, z.B.

Frage: „Stichtag“

Programmvorschlag: **Today()**

Das Jahr ist: Today()->year.

IX. Hilfen zur Übersichtlichkeit

Items rufen sich in ihren Bestimmungsfunktionen und den Bedingungen gegenseitig mit ihren **names** auf.

Im Dialogprogramm ExpertItems stehen die Items auf dem Arbeitsblatt (rechts) linear untereinander. Da jedes von ihnen zahlreiche Eingabefelder untereinander enthält, können nur wenige auf dem Arbeitsfeld (rechts) sichtbar sein.

Diesen Nachteil gleicht zuerst einmal das Navigationfeld (Treeview, Bäumchen) auf der linken Seite oben aus, welches eine Übersicht, in der die Items zeilenweise untereinander stehen, liefert, so dass man schnell von einem Item zum anderen navigieren kann.

Doch bei größeren Programmen ist auch hier bald die Grenze der Übersichtlichkeit erreicht. Man ist deshalb gezwungen, die Items in Gruppen zusammenzufassen und mehreren Items einen gemeinsamen Gruppennamen zuzuweisen. Dadurch kann man im Navigationsfeld viele Item zu einem ÜberItem zusammengefasst – auf und zuklappen – und über zahlreiche Stufen eine Übersicht über beliebig viele Items gewinnen (s.u. **Gruppierungsknoten**).

Ein zweites Problem besteht in der individuellen Vielfalt: In den Programmen ist von vielen Menschen, Sachen, Versorgungen u.s.w. die Rede, die jeweils in ihrer Grundstruktur übereinstimmen, in ihren Merkmalen (Alter, Größe, Anspruchshöhe u.s.w.) aber verschieden sind. Wenn sie durch Items abgebildet werden, müssen diese einerseits die gemeinsame Struktur beschreiben, andererseits aber auch das jeweilige Individuum oder den individuellen Gegenstand identifizieren.

Auf das Ordnungsproblem anwortet die Möglichkeit zum

1. Ausblenden von Leereingaben

Über das Hängemenü können mit **Bearbeiten/Leereingaben ausblenden** im Arbeitsblatt die leeren Felder ausgeblendet werden. In der Kopfzeile befindet sich dafür auch ein Button (+-) mit der gleichen Funktion. Am bequemsten dürfte der Zugang über die rechte Maustaste sein. Dadurch passen viel mehr Items in das Arbeitsblatt (rechte Seite). Will man ein solches Feld bearbeiten, muss man es wieder einblenden. Der Gewinn an Übersichtlichkeit wird schnell erreicht.

2. Umsetzen von Items

Man kann zusammengehörige Items dann, wenn sie durch Zufall weiter voneinander entfernt eingetragen wurden, zusammenführen. Hierzu muss zuerst der Name des umzusetzende Items angeklickt und über das Hängemenü mit **Bearbeiten/Listenelement löschen** vorläufig gelöscht werden. Sodann wird der Name des Items, vor dem das gelöschte eingefügt werden soll, angeklickt und sodann **Bearbeiten/Listenelement einfügen** gewählt. Dadurch wird das zuletzt gelöschte Item (auf der gleichen Ebene) vor dem markierten Item eingefügt.

Wenn zusammenhängende Items beieinander stehen, lassen sie sich leichter finden.

3. Einfügung eines Gruppierungsknotens

Durch Einfügung eines **Gruppierungsknoten** kann nachträglich, wenn ein wachsendes Programm unübersichtlich geworden ist, eine zusammenhängende Reihe von **Items** unter einem Gruppennamen zusammengefasst werden, sodass im **Navigationsfeld** sie durch Klick zum Erscheinen und zum Wiederverschwinden gebracht werden können.

Weil der **Gruppierungsknoten** auf die Datenstuktur keinen Einfluss hat, muss dessen Name immer in runden Klammern gesetzt sein, z.B. *(Gruppe)*. Im **Navigationsfenster** kann man dadurch die **Gruppierungsknotens** von den **Einzelknoten** (s.u.) unterscheiden – wichtig wegen der verschiedenen Adressierung: Wird eine Ebene mit **Gruppierungsknoten** eingefügt, dann ändert das nichts an der Addressierung von aufgerufenen Items.

Wählt man **Gruppierungsknoten** im Untermenü **Knoten**, dann entsteht im **Navigationsfenster** ein **Topic** und alle darunter liegenden Items werden um eine Stufe nach rechts gerückt.

Um die Items, die nicht zu der Gruppe gehören, wieder auf das alte Niveau zu heben, muss der Name des ersten Items, welches nicht zu der neuen Gruppe gehört, angeklickt und über das Hängemenü mit **Bearbeiten/Listenelement einfügen** ein neues Item eingefügt werden, dessen Namen zu löschen bzw. leer zu lassen ist, damit es das Ende der Gruppe markiert.

(Will man mit „neues Listenelement einfügen“ ein neues Item einfügen, so bewirkt, das kurzfristig das Ausscheiden der darunterliegenden Items aus der Gruppe, bis der neue Itemname eingefügt ist. Will man ein neues Item am Ende der Gruppe anfügen, dann muss man dasselbe in das freie Endfeld eintragen, weil man in ein leeres Feld kein weiteres leeres Feld einfügen kann. Nach Eingabe des

neuen Item-Namens muss dann die untere Begrenzung mit „neues Listenelement einfügen“ wieder eingefügt werden.[3]

Gruppierungsknotens können auch in **Gruppierungsknoten** Bereiche eingefügt werden, sodass mehrstufige Gruppierungen entstehen.

X. Ordnung von Items durch Listen und Einzelknoten

Das Problem der individuellen Vielfalt wird durch die **Listen (lists)** und **Einzelknoten** bewältigt. Jede Liste enthält gleichartige Elemente, die durch ihre Indizes – welche in eckige Klammern geschrieben werden - unterschieden werden. Die Verwendung von Listen ermöglicht beliebig viele Programmdurchgänge für beliebig viele Listenelemente. Beispiel: Der Unterhalt soll für mehrere Kinder bestimmt werden. Jedes Kind wird durch ein Listenelement repräsentiert.

Listen und **Einzelknoten** sind Gruppen-Items oder Oberbegriffe für mehrere Items. Sie helfen die ihnen zugeordneten Items zu finden, werden aber ebenso wie echte Items mit der Eingabe ihres Namens erzeugt. Im Menü den Typen jedoch wird ihnen mit **Knoten** die Ordnungsaufgabe zugewiesen.

Wenn im Typenmenü **Knoten** gewählt wurde, öffnet sich zur weiteren Bestimmung des Typs das Untermenü der **Knoten:**

Einzelknoten

Nummerierte Liste

Zugeordnete Liste

Namensliste

(Gruppierungsknoten) (s.o. IX.3.)

Gruppierungsknoten dienen ausschließlich der **Ordung von Items** im Navigationsfenster. Dort erscheinen sie wie **Einzelknoten,** ihre UnterItems können wie bei diesen auf- und zugeklappt werden. Doch ändern sie nicht die Datenstruktur. Bei der Addressierung bleiben sie unberücksichtigt. Daher können Gruppierungsknoten problemlos nachträglich eingefügt werden, sobald die Vielzahl der Items unübersichtlich wird.

Für alle **Listen** gilt, dass ihre Elemente wieder **Listen** oder aber **Einzelknoten** sind und nie einzelne Items. Deshalb wiederholt sich auch bei jeder Liste die Frage nach den **Knoten. (**Nur die **Gruppierungsknoten** wird dann nicht mehr angeboten.)

1. Namensliste

Die einfachste Form ist die **Namensliste.** Die einzelnen Elemente einer **Namensliste** werden hier durch ihren Namen unterschieden, welcher den Index darstellt, welcher beim Aufruf in eckige Klammern geschrieben wird.: z.B. *Listenname[Elementname]*. Eine solche Liste kann nicht als ganzes aufgerufen werden, weil nicht durchgezählt werden kann. Nur mit der speziellen Funktionen **Listenelemente zählen** und **Listenelemente addieren (s.u.)** kann ihre Gesamtheit erfasst werden.

[3] Die Anwendungsregeln sind technisch bedingt. Bisher wurde noch kein Weg gefunden, sie zu vereinfachen.

Sie eignet sich vor allem dazu, ein Element in einem Arbeitsgang anzulegen und etwa durch eine Frage zu bestimmen. Als Elementname kann jeder Text verwendet werden, der in dem aufrufenden Item erzeugt wird, vorausgesetzt, er unterscheidet sich von einem bereits Erzeugten – wenn denn nicht die diesem zugeordeten Werte verwendet werden sollen.

Nun können die Elementnamen aus Eingaben stammen oder von Anfang an im Programm festgelegt sein. Deshalb fragt das Programm

Namen in der Liste können ersetzt werden?

Die Frage ist zu bejahen, wenn die Namen im Programmlauf durch eine Eingabefunktion festgelegt wird, wenn als das Programm z.B. nach den Namen der Kinder fragt und diese als Index der **Namensliste** verwendet. Denn dann kann im Programmlauf der Name auch einmal geändert werden – und sei es nur, weil sich ein Schreibfehler eingeschlichen hat. Wenn der Name sich jedoch nicht ändern kann, weil er programmiert wurde, kann *nein* eingegeben werden.

Weil die **Namensliste** immer auch mehrstufig sein kann, folgt anschließend die Menü-Frage

Einzelknoten

Nummerierte Liste

Zugeordnete Liste

Namensliste

Wird **Nummerierte Liste, Zugeordnete Liste** oder **Namensliste** gewählt, so erhalten diese Listen keine eigenen Namen. Vielmehr hat die erzeugte neue Liste den dort angegebenen Inhalt. Die ursprüngliche Liste liefert lediglich den ersten Index.

Ist die Liste nicht mehrstufig (mehrere Indizes), muss **Einzelknoten** gewählt werden, selbst wenn nur ein einziges Unter-Item gebraucht wird (weil der Name ja bereits im Index steht).

2. Nummerierte Liste

Bei der **Nummerierte Liste** werden die einzelnen Items abgezählt und durch ihre laufende Nummer, welche als Index dient und in eckigen Klammern gesetzt wird, unterschieden. Beginn der Zählung ist immer die Null, das erste Element der Liste ist daher immer <Name der Liste>*[0]*, das dritte Element <Name der Liste>*[2]*. Die Zählung ermöglicht es, die Liste als ganzes zu bearbeiten, wovon die Typen **Listenelemente zählen, Listenelemente addieren** und **Kumulative Addition** Gebrauch machen (s.u.).

Nach der Wahl von **Nummerierte Liste** erscheint als erste Frage:

Item, welches den Namen des Elements enthält

Durch Einfügung dieses Namens erhält die **Nummerierte Liste** zugleich die Qualität einer **Namensliste**, denn dann kann ein Element der Liste nicht nur mit seiner Nummer, sondern auch durch diesen Namen adressiert werden (vorausgesetzt natürlich, dass dieser Name auch als Item des Elements eingetragen wird – wenn er fehlt, kommt es daher zu einer Fehlermeldung).

Ist ein **Item, welches den Namen des Elements enthält** eingetragen, kann auf die Liste mit **Zugeordnete Liste** (s.u. (3)) referenziert werden.

Wird **kein Name** eingegeben, dann folgt die Frage:

Bedingung, dass this_list[i] (oder auch l[i]) der erste Knoten ohne Daten ist

Sonst wird nämlich das Ende der Liste dadurch bestimmt, dass der Name „“ lautet. Fehlt auch ein eigenes Namensfeld, dann muss eine Bedingung für das Ende der Liste angegeben werden, wobei der Zähler mit „i“ bezeichnet wird und das Element mit „this_list[i]“ oder „l[i]“.

ACHTUNG! Diese Bedingung ist NICHT auf derselben Ebene wie die Liste selbst. Wenn darin auf Items Bezug genommen wird, welche sich auf derselben Ebene wie die Liste befinden, so muss vor ihren Namen „up.“ gesetzt werden!

Wird ein **Item, welches den Namen des Elements enthält** benannt, dann kann **Nummerierte Liste, Zugeordnete Liste** und **Namensliste** nicht mehr eingegeben werden. Deshalb fällt die Frage **Einzelknoten, ….** aus und es folgt anschließend die Frage nach den Items des Elements, beginnend mit der Frage nach dem Namens-Item, für welches der vorher eingegebene Name vorgeschlagen wird. Außerdem wird die Zählung dafür verwendet, eine Frage vom Typ *Name (x.),* wobei *x.* die Zählung darstellt, dem Programmierer vorzuschlagen. Wird der Vorschlag übernommen oder sonst ein entsprechendes Item mit dem Typ **Text** eingegeben, dann folgt auch hier (wie bei der **Namensliste**) die Frage

Item enthält einen ersetzbaren Namen?,

weil auch die **Nummerierte Liste** mit **Item, welches den Namen des Elements enthält** letztlich zugleich eine **Namensliste** darstellt.

(Als Name kann auch eine **Variable**, nämlich eine Textvariable, verwendet werden, wodurch es möglich wird, ein ExpertItems-Programm in ein C- oder C++-Programm einzubetten, indem man nämlich den Namens-Variablen einer Nummernliste die Elemente einer C-Liste zuweist und diese dann auf eine zugeordneten Liste überträgt.)

Der Name dient auch dazu, eine nummerierte Liste abzuschließen: Werden mit „Listenelemente Zählen“ die Namen der Elemente eingetragen, dann bestimmt eine Leereingabe das Ende der Liste. Wird jedoch kein Name festgelegt, dann muss eine andere Bedingung das Listenende bestimmen. Hiernach fragt das Programm (solange kein Name eingetragen ist):

…Bedingung, dass this_list[i] (oder auch l[i]) der erste Knoten ohne Daten ist

Hier kann jede Bedingung eingetragen werden, welche von ‚i‘ abhängt (vgl. auch die booleschen Item oben V.).

3. Zugeordnete Liste

Wird der Listentyp **Zugeordnete Liste** gewählt, folgt die Frage nach der referenzierten Liste

Hauptliste mit Namen?

Hier kommt jede **Nummerierte Liste** in Betracht, welche zugleich ein **Item, welches den Namen des Elements enthält** beinhaltet. Dessen Name ist hier einzugeben.

4. mehrstufige Listen

Alle Listen können mehrstufig sein. Zugleich eine **Nummerierte Liste** und eine **Namensliste** kann aber nur die der letzten Stufe sein.

5. Adressierung in Listen

a) Aressierung eines Items in einer Liste von der Hauptebene aus

Soll von der Hauptebene, auf welcher sich die Liste befindet, ein Item in einer einstufigen Liste aufgerufen werden, dann muss das Listenelement durch Listennamen und Index (in eckigen Klammern) und, durch einen Punkt getrennt, der Name des Unter-Items angegeben werden, z.B. bei einer **Nummerierte Liste** *Listenname[Indexzahl].Itemname*, oder bei einer **Namensliste** *Listenname[Indexname].Itemname*. Ist die Liste zweistufig, so sind beide Indizes hintereinander zu setzen, z.B. bei zwei **Nummerierten Listen** *Listenname[Indexzahl1][Indexzahl2].Itemname*, oder bei zwei **Namenslisten** *Listenname[Indexname1][Indexname2].Itemname*.

b) Adressierung eines Items der Hauptebene von einem Listenelement aus: up.

Soll ein Item auf der Elementebene einer einstufigen Liste ein Item auf der Hauptebene aufrufen, dann muss zweimal „up" (s.o. VII.2.c) verwendet werden, weil man zuerst auf die Stufe des **Einzelknoten** aufsteigen muss und danach weiter auf die Stufe der Liste, z.B. „up.up.Itemname". Ist die Liste mehrstufig, dann kommt für jede weitere Stufe ein „up." hinzu. Ebenso kann jedoch auch die fragliche Ebene addressiert werden, und zwar mit **upto(Name des Knotens).** Als Argument ist der Name des Knotens einzutragen, dessen Element aufgerufen werden soll.

6. Sonderzeichen

Mit **XI_number** kann in einem zu einem Listenelement gehörigen Item der numerische Index des Elements aufgerufen werden. Bei einer zweistufigen Liste sind der letzte Index ***XI_number*** und der entferntere ***up.XI_number*.**

In einer **Zugeordnete Liste** kann der Name eines Elements in diesem mit **XI_GetElementName(XI_number*)*** aufgerufen werden, allgemein ist der Name eines Elements i einer Liste *<Listenname>: <Listenname>.**XI_GetElementName(i),*** wobei „i" der numerische Index des Elements ist.

Mit **XI_name** kann man den Namen eines Elements in demselben direkt aufrufen und etwa die Bedingung verwenden XI_name ==“der Anbietende".

7. Einzelknoten

wird in den Listen gebraucht, weil diese grundsätzlich nur Datensätze, nicht einzelne Items, auflisten. Ein Item, das zu einer Liste gehört, muss deshalb immer über einen Einzelknoten erchlossen werden. Wird dieser Typ gewählt, folgt die

a) Eingabe der Unter-Items

Dafür ermöglicht das Programm die Neueingabe aller diesem **Knoten** zugeordneten Items:

z.B. **Name d. 1. Items d. yyyyyy::xxxx:**

b) Adressierung der Unter-Items

Während die **Gruppierungsliste** bei Addressierung der Items unberücksichtigt bleibt, muss beim Aufruf eines Items in einem **Einzelknoten** dem Namen des Items der Name des **Einzelknoten,** getrennt durch einen Punkt **(.)** vorangestellt werden, z.B. *Invalidenrente.Hoehe*, wenn *Invalidenrente* ein **Einzelknoten** ist und *Hoehe* ein diesem zugehöriges Item.

c) Adressierung in einem Unter-Item

Will man aus dem Unter-Item heraus ein Item in der übergeordneten Ebene aufrufen, so genügt dazu nicht – dessen Name. Vielmehr muss auch darauf hingewiesen werden, dass sich das Item nicht im

Bereich des **Einzelknoten,** sondern eine Ebene höher befindet. Zu diesem Zweck wird dem Namen des Items die Silbe **„up“** vorangestellt, wieder getrennt durch einen Punkt (.),

z.B. *up.Ausgleichswert*, wenn das Item *Ausgleichswert* von einem Item aufgerufen wird, welches sich eine Stufe unterhalb der Ebene von *Ausgleichswert* befindet, oder (Kombination von herauf und herunter:)

up.casusB.Bezeichnung1, wenn das Item *Bezeichnung1* sich unter einem **Einzelknoten** *casusB* befindet, und von einem Item eine Stufe unterhalb der Ebene, auf welcher sich *casusB* befindet, aufgerufen wird.

8. Gruppierungsknoten dienen **nur der Ordnung der Items, s.o. IX.3.**

XI. Weitere Verwendung der Listen (Menüpunkt other)

Die unmittelbare Verwendung einer Liste durch andere Items ist nur bei einer **Namensliste möglich**. Sonst müssen sie durch spezielle Items erschlossen werden (soweit man nicht auf die Programmierung in C oder C++ zurückgeht).

Diese weiteren Items werden im Typenmenü angeboten über den Menüpunkt **sonst**.

Wird **sonst** gewählt, dann öffnet sich ein **Untermenü**

Listenelemente zählen

Listenelemente addieren

Kumulative Addition

Kolonne von Geldbeträgen

Eingerückte Kolonne von Geldbeträgen

Die ersten beiden,

Listenelemente zählen,

Listenelemente addieren

erschließen Listen, die der Letztanwender durch seine Einträge erzeugt hat, während die übrigen einen Listenersatz für vom Experten strukturierte Additionskolonnen darstellen und anschließend dargestellt werden (s.u. X.). Alle diese Funktionen ermöglichen es, alle Elemente einer Liste in Reihe nacheinander zu bearbeiten. Dieses ‚Nacheinander‘ kann so im Vordergrund stehen, dass das eigentliche Ergebnis, z.B. die Anzahl bei **Listenelemente zählen** nicht interessiert und deshalb weder ausgegeben noch sonst verwendet wird.

1. Listenelemente zählen

Vielfach möchte man zu Beginn der Berechnung erst einmal alle Beteiligten feststellen. Hierzu dient **Listenelemente zählen.** Es fragt

Bei einer **Nummerierte Liste** mit **Item, welches den Namen des Elements enthält** die zugehörigen Namen ab und ermittelt dabei, wie viele Elemente die Liste haben soll.

Als Argument muss nur der Name der Liste gemeldet werden - welche getrennt von diesem Item als **Nummerierte Liste** mit **Item, welches den Namen des Elements enthält** anzulegen ist:

" **Name der Liste, deren Elemente gezählt oder addiert werden sollen**"

In der Liste **muss** mit **Item, welches den Namen des Elements enthält** ein Namens-Item schon vorhanden sein. Die Abfrage der einzelnen Elemente braucht sich aber nicht auf den Namen zu beschränken (auch wenn das der Hauptanwendungsfall **Listenelemente zählen**

sein dürfte**)**. Man kann im Feld **Nach Wertbestimmung** des Namens-Items weitere Items des jeweiligen Elements aufrufen. Dann folgen auf die Namenseingabe die entsprechenden Fragen.

z.B. **Namensfrage** *Name des Kindes (1.)?*, **Nach Wertbestimmung:** *Alter_des_Kindes();*. Dann wird nach Eingabe eines Namens das Alter abgefragt.

Die errechnete Anzahl muss vom Programm nicht gemeldet werden. Wenn als **Bedingung für die Ausgabe** *false* eingetragen ist, wird zwar der Name der Kinder abgefragt, ihre Anzahl aber nicht gemeldet. Ebenso kann bei der Zählung selbst unterschieden werden: Bei der Frage „**Bedingung für die Zählung des Elements mit dem Index** i" kann die entsprechende Bedingung eingetragen werden.

Achtung: Der Praktiker will die Eintragung in Listen oft ändern, insbesondere leicht ein Listenelement löschen, ohne dass ihm der Rest der Liste verloren geht. Das wird dadurch gesichert, dass bei der Abfrage des Namens in das Fragefeld vor den Text der Frage „Empty_input_list_end()", eingefügt wird, z.B. kann in ein Eingabefeld „Frage" eingetragen werden: Empty_input_list_end(), „Name eines (weiteren) Kindes?"

2. Listenelemente addieren

Die meisten Arbeiten an einer Liste bestehen in einer selektiven Addition oder lassen sich als solche darstellen. **Listenelemente addieren** bedeutet, dass bestimmte Elemente aus einer **nummerierten Liste** aufaddiert werden sollen. Zuerst muss allerdings – mit einem Menü - der Typ der Werte festgelegt werden, die addiert werden sollen. Angeboten werden

Ganzzahl

Kommazahl

Geldwert

Prozentsatz

Text

Mehrspaltige Kommazahl

Mehrspaltiger Geldbetrag

Ist die Auswahl erfolgt, wird nach der Liste gefragt, deren Werte zu summieren sind:

Name der Liste, deren Elemente gezählt oder addiert werden sollen,

nach dem zu addierenden Element:

Name des aufzusummierenden Items,

nach der Bedingung, unter welcher diese Addition stattfinden soll:

Bedingung für die Addition (Index ist 'i')

sowie nach dem resultierenden Datentyp, welcher aus einem Menü ausgewählt wird, also den Datentyp der Summe.

Werden Ja/Nein-Items oder ganze Zahlen addiert, ist das Resultat eine **Ganzzahl**, bei Geldbeträgen wieder solche u.s.w. Auch **Texte** können addiert werden. Das führt zur Verkettung der Element-Texte.

Die weitere Frage

Als Kolonne ausgeben

ermöglicht es, in der Ausgabe alle Additionsposten in einer Rechnungskolonne untereinander zu schreiben, welche durch einen Schlussstrich abgeschlossen wird und an welche die Summe aller Einzelposten anschließt. In der Kolonne werden die Listenelemente mit ihren Namen aufgeführt. Zusätzlich kann eine Zählung erfolgen, welche den Namen vorangestellt wird, z.B. „1. Anton“. Hierfür schlägt das Programm einen vorangestellten Text vor, der auch geändert werden kann.

In diesen Zusammenhang gehört auch die

3. Kumulative Addition.

Sie steht nicht wie **Listenelemente addieren** neben der Liste, sondern ist ein weiteres Item des jeweiligen Listenelements. In diesem Listenelement werden die Werte eines anderen Items des Elements stufenweise aufaddiert (einschließlich des entsprechenden Items des Listenelements). Dieses andere Item wird mit

Name des aufzusummierenden Items

abgefragt.

XII. Spezielle Additionskolonnen

Erinnern wir uns: **Listenelemente addieren** kann in einer Kolonne mit Schlussstrich erfolgen. Diese Kolonne entsteht durch Eingabe der einzelnen Elemente einer Liste und kann daher beliebig lang sein. Eine Kolonne wird aber viel häufiger gebraucht, um bestimmte vordefinierte Geldbeträge – wie in einem Fragebogen - aufzuaddieren. Zu diesem Zweck wurde

1. Kolonne von Geldbeträgen

entwickelt. Eine Liste braucht hier nicht angelegt zu werden. Für den Zweck der Addition von Geldbeträgen ist hier ein besonderer Typ geschaffen worden, welcher es ermöglicht, eine Additionskolonne zu gestalten und dabei die Bezeichnungen und die Beträge frei zu bestimmen.

Da hier auch eine mehrspaltige Berechnung möglich ist, erscheint zuerst das Menü

Geldwert

Geldwert, mehrspaltig

Die mehrspaltige Berechnung wird unter XI. zu erörtern sein. Deshalb wählen wir **Geldwert**.

Danach muss geklärt werden, unter welcher Bedingung der Ausdruck (die Ausgabe) erfolgen soll. Ein eigenes Feld zur Eingabe der jeweiligen Bedingung würde das Item sehr aufblähen und – falls immer dieselbe Bedingung einzutragen ist - unnötige Arbeit verursachen und das Item unübersichtlich machen. Deshalb ist ein Menü vorgeschaltet, das die Anwahl der einfachen Bedingungen vorwegnimmt:

Nullwerte unterdrücken

spezielle Bedingungen

normal

Nullwerte unterdrücken bewirkt, dass nur solche Posten ausgegeben werden, für die ein Geldbetrag eingetragen wurde, dass also Nullpositionen ausgespart werden.

Spezielle Bedingungen ist der allgemeine Fall, bei welchem für jeden Posten gesondert abgefragt wird, unter welchen Voraussetzungen er gemeldet werden soll und

normal bedeutet, dass jeder Posten ausgegeben wird, gleichgültig, welchen Wert er hat. Auch andere Bedingungen werden dann nicht geprüft.

Danach werden die einzelnen Posten abgefragt;

Name der zu addierenden Position

Hier ist die Kennzeichnung einzugeben. Dafür bestehen zwei Möglichkeiten: entweder wird der Text selbst mit Anführungszeichen eingegeben, z.B. ‚„Einkommen“', oder es wird auf ein Item des Typs **Text** Bezug genommen, z.B.: *einkommen*.

Nur wenn **Name der zu addierenden Position** einen Eintrag erhält, erscheinen die folgenden Fragen (ggf. wird die Eingabe beliebig oft fortgesetzt, d.h. die Leereingabe ist hier Abbruchbedingung). Optional (nur wenn **spezielle Bedingungen** gewählt wurde) folgt

Bedingung für die Verwendung des Werts x (z.B. "x!=0")

Hier ist die Bedingung dafür einzutragen, dass das aufgerufene Unter-Item in der Addition berücksichtigt wird. Wenn auf den Wert Bezug genommen werden soll, ist für diesen Wert in der Bedingung „x“ zu setzen. In jedem Fall folgt die Frage

Geldbetrag für die Addition

Hier ist der zu addierende Betrag einzufügen – regelmäßig durch Angabe des betreffenden Items vom Typ **Geldbetrag**.

Anschließend wird die Frage „**Name der zu addierenden Position**“ wiederholt, bis das Eingabefeld leer gelassen wird.

Die folgenden Eingabefelder **Überschrift nach der Bestimmung des Itemwerts, Standard-Ergebnistext, Bedingung für die Ausgabe** und **Nach Wertbestimmung** haben übliche Bedeutung. **Bedingung für die Ausgabe** bezieht sich auf die ganze Ausgabe.

Achtung! Die Kolonne wird erst **nach** der Bestimmung der einzelnen Werte ausgegeben. Oberhalb der Kolonne befinden sich deshalb die Fragen nach den einzelnen Beträgen!

2. Eingerückte Kolonne von Geldbeträgen

ermöglicht dem Anwender, in gleicher Weise wie mit **Kolonne von Geldbeträgen** eine Additionskolonne zu konstruieren, jedoch ist diese – als „Unterkolonne“ nach links eingerückt. Das Ergebnis kann dann wiederum Einzelposten der Hauptkolonne sein.

XIII. Mehrspaltig rechnen (Vektoren)

In der anwaltlichen und gerichtlichen Praxis sind oft mehrere Standpunkte zu beachten. Das Zugewinnausgleichsprogramm von WinFam (Familienrechtliche Berechnungen bei C.H. Beck) ermöglicht seit langem, in bis zu vier Spalten nebeneinander zu rechnen, ebenso die Berechnung des Nachlasses im Winerb-Programm (Erbrechtliche Berechnungen bei C.H. Beck). Damit die Umstellung auf ExpertItems zu keiner Leistungseinschränkung bei WinFam führt, muss ExpertItems auch die mehrspaltige Berechnung ermöglichen.

Das Typen-Menü liefert unter **sonst** mehrspaltige, also Vektor-Typen:

Mehrspaltige Kommazahl

Mehrspaltiger Geldbetrag

Mehrspaltiger Text

Der Quelltext des neuen Zugewinnausgleichsprogramms zeigt, wie in ExpertItems mit Vektoren gerechnet werden kann. Ein Kernproblem besteht darin, dass sich die Berechnung verzweigen kann, und dass diese Verzweigung in verschiedenen Spalten unterschiedlich verlaufen kann. Beim Zugewinnausgleich erfolgt eine Verzweigung nur bei der Feststellung, wer ausgleichsberechtigt ist. Es werden hier alle Zweige berechnet.

Wenn ein Wert mehrspaltig eingegeben wird, stimmen die Werte teilweise überein. Dann aber ist die Standardvorgabe „0“ wenig sinnvoll. Hier kann in das Feld mit dem Eingabevorschlag eine spezielle Funktion eingegeben werden.

Default_in_vector_since(x, 0. , "<version>")

Das erste Argument „x“ bedeutet, dass der Eingabewert eines etwa vorangehenden Feldes als Vorschlag verwendet werden soll, das zweite Argument ist der Vorschlag, wenn kein Feld vorangeht und das dritte Argument gibt an, ab welcher Programmversion diese Regelung gelten soll.

Das **Rechnen** mit mehrspaltigen Zahlen (Vektoren) ist einfach: Wenn die Vektoren mit der gleichen Spaltenzahl definiert wurden, können sie einfach addiert und subtrahiert werden. Das ergibt wieder einen Vektor. Wird eine Einzelne Zahl zu dem Vektor hinzuaddiert oder subtrahiert, dann entsteht wieder ein Vektor, bei welchem jede einzelne Zahl in gleicher Weise verändert wurde, ebenso, wenn mit einer Einzelnen Zahl multipliziert oder durch dieselbe dividiert wurde.

Auch die mehrspaltige Darstellung von mehreren verschiedenen Rechenläufen des (für die Unterhaltsberechnung) wurde inzwischen realisiert:

XIV. Funktionen und Variablen

ExpertItems verwendet die Programmiersprache C++. Mit C++ geschriebene Programme können in ExperItems eingebettet werden, umgekehrt können in ein C- oder C++ Programm auch ExpertItems-Codierungen eingefügt werden – wie das bei „WinFam“ und „Winerb“ auch tatsächlich geschieht.

1. Funktion

Das Quasi-Item **Funktion** hat einem Namen wie andere Items auch. Außerdem wird ebenso wie bei den echten Items ein Datentyp als Ergebnis abgefragt. Jedoch kann, anders als bei einem echten Item, die **Bestimmung des Itemwerts wiederholt** werden. Hierfür sind zwei Eingabefelder maßgebend:

Liste der Funktionsargumente

Hier ist einzutragen, was bei Aufruf der **Funktion** ihr als Information mitgegeben werden muss, damit sie ihre Berechnung durchführen kann. Dabei müssen alle Argumente mit ihrem Typ und einem Namen, der sie in dem folgenden

Funktionskörper

vertritt, eingetragen werden. Letzteres ist ein kleines C oder C++-Programm, welches sein Ergebnis mit *return xxx;* exportiert.

z.B.:

Liste der Funktionsargumente: *double a, double b, int c*

Funktionskörper: *return (a+b)*c;*

In einem **Item** wird eine **function** in einer Bestimmungsfunktion mit den Argumenten in der Klammer verwendet,

z.B.

Berechnung(add1, add2, faktor)

Wenn *add1* und *add2* Items vom Typ **Kommazahl** (in C++: „double“) und *faktor* ein Item vom Typ **Ganzzahl** (in C++: „Ganzzahl“) ist und *Berechnung* eine **Funktion** nach dem vorigen Beispiel.

Funktionen dürfen keine Fragen enthalten, weil diese nicht sachgerecht protokolliert werden können. Auch kann ihre Verwendung die Fehlersuche erschweren, weil die Suche nach der Herkunft eines Itemwerts dort abbricht.

2. Variable

Auch dieser Typ wird über die Zwischenstufe **sonst** erreicht. Er hat seinerseits einen bestimmten **Datentyp** wie **Ganzzahl, Text** etc. und man kann ihren aktuellen Wert wie den eines anderen Items zur Bestimmung eines Items verwenden.

Um der **Variablen** einen Wert zuzuweisen, ist eine Zuweisung nach C oder C++ erforderlich, also z.B.

*variable = Item1 + Item2*75;.*

Das „;“ am Ende darf nicht vergessen werden! Erfolgt die Zuweisung im Eingabefeld einer Bedingung einem Bedingungsfeld, so kann die Zuweisung auch nur durch Komma getrennt vor die Bedingung gesetzt werden. Diese Zuweisung kann in jedem anderen Item erfolgen. Allerdings muss die **Variable** wie jedes Item mit ihrem korrekten Pfad aufgerufen werden.

Variablen sollten nicht verwendet werden, wo Items möglich sind. Bei der Fehlersuche sind sie ebenfalls störend.

XV. Transparenzfunktionen

ExpertItems ist auf höchstmögliche Transparenz für den Letztanwender angelegt.

1. programmierte Transparenz

Es ist allerdings in erster Linie Aufgabe des Experten, das erzeugte Programm so zu gestalten, dass die Zusammenhänge transparent sind. Deshalb muss er dafür Sorge tragen, das bei logischem Programmieren alle für die Schlussfolgerung maßgeblichen Voraussetzungen und auch alle Zwischenschritte vom Programm durch Ausgabefunktionen (s.o. Kap. V.) gemeldet werden. Da die logischen Stationen regelmäßig einzelne Items sind, sollte deshalb deren Wert durch Ausgabefunktionen (z.B. String_output()) dem Letztanwender bekannt gemacht werden.

2. Unterstützung transparenter Rechenwege

Wenn der Experte Ergebnisse mit den vier Grundrechenarten erzeugt, dann unterstützt ihn das Programm, indem es die Ausgabe des konkreten Rechenwerks vorschlägt. Oben ist das unter VI.5. dargestellt. Es handelt sich aber nicht um ein starres System. Vielmehr schlägt, wenn ein solches Rechenwerk als Bestimmungsfunktion für ein **besonderes Ergebnis** oder eine **Rechenfunktion** eine Standardausgabetext vor, welcher dann das konkrete Rechenergebnis darstellt.

Wenn z.B. ein Item mit dem Namen *Gattenunterhalt* den **Rechenausdruck** *(Manneseinkommen-Fraueneinkommen)*3/7* enthält,

dann schlägt das Programm als Standardausgabe vor:

„(**string**) *Gattenunterhalt: („+Manneseinkommen+“ – „+Fraueneinkommen+“)*3/7“.*

Wird dieser Vorschlag übernommen, dann wird ein Programm erzeugt, welches dann, wenn z.B. ein Manneseinkommen von 3000 € und ein Fraueneinkommen 1600 € eingegeben wurde, folgendes Ergebnis meldet:

*Gattenunterhalt: (3000-1600)*3/7 600 EUR*

Der programmierende Experte kann aber auch eingreifen, beispielsweise hinter *3/7* noch ein Gleichheitszeichen = setzen. Dann erscheint

*Gattenunterhalt: (3000-1600)*3/7= 600 EUR*

3. Transparenzfunktion auch für den Letztanwender

Die Programmierung mit ExpertItems ermöglicht es aber auch dem Letztanwender, sich selbstständig durch Einblick in den Programmlauf das Ergebnis zu erklären und damit auch mögliche Fehler selbst aufzuklären. Das soll die Autonomie des Programm-Anwenders stärken. Doch auch die Arbeit des programmierenden Experten wird damit unterstützt (s. im Anhang: Fehlersuche). In dem Hängemenü (Pull-Down-Menü) „Optionen“ befindet sich dafür ein Unterpunkt „Menüpunkte zur Berechnungstransparenz ein/aus“. Wird die Option eingeschaltet, dann erscheint die Meldung:

Über die rechte Maustaste werden jetzt im Berechnungsfenster zusätzliche Menüpunkte eingeblendet. Mit diesen kann man sich in den Programmmodulen, die mit der Programmiermethode ExpertItems erstellt wurden, Details zur internen Datenverarbeitung einblenden lassen - sowohl Zwischenergebnisse als auch C++-Programmteile. Sie sind über die rechte Maustaste im Berechnungsfenster zu erreichen.

Sollen bei diesen Zusatzausgaben technische Hilfetexte zur Bedeutung dieser Zusatzausgaben statt zum Berechnungsinhalt gezeigt werden?

Ja Nein

Die Transparenzfunktion als solche kann auch durch Hilfetexte unterstützt werden. Wenn der Anwender mit ihr noch nicht vertraut ist, wird er deshalb die abschließende Frage bejahen, sonst ist es wohl günstiger, wenn die Hilfefenster sich an dem Thema des erzeugten Programms beschäftigen. Dann sollte mit <n> geantwortet werden.

In jedem Fall wirkt sich die nun eingeschaltete Transparenzfunktion nur aus, wenn mit der rechten Maustaste an irgendeiner Stelle des Programms Transparenzinformationen abgefragt werden. Dabei ist nur die rechte Maustaste zu verwenden. Es darf nicht vorher in ein Feld „hineingeklickt" werden.

Die Transparenzfunktion verschafft rein technisch einen Zugang zu den verwendeten Items, und erläutert sie nicht.

Wenn an beliebiger Stelle im Programmlauf die rechte Maustaste betätigt wird, erscheint der Name des zugehörigen Items mit der Frage, ob dieses gezeigt werden soll. Auf diese Weise können allerdings nur Items unmittelbar erreicht werden, welche auf dem Bildschirm eine Ausgabe liefern, also eine Frage, eine Überschrift, eine Standardausgabe oder sonst eine Meldung, die in dem Feld **Nach Wertbestimmung** eingetragen wurde. Andere Items können nur indirekt erreicht werden, s.u.

Bejahendenfalls wird eine Reihe von Optionen angeboten:

a) Auswertungsgrund zeigen

Der Auswertungsgrund ist das Item, in dessen – im konkreten Fall verwendeter – Bestimmungsfunktion dieses Item gebraucht und deshalb aufgerufen wurde. Wird der Auswertungsgrund verlangt, dann tauchen oberhalb des betreffenden Items die anfragenden Items in der jeweiligen Position im Programmlauf auf mit Fragezeichen (z.B. *Tenor.TBewilligung?)* sowie weiter unten mit dem jeweiligen Ergebnis (z.B. *Tenor.TBewilligung=true*). Dieselben Einträge finden sich dann auch in dem Orientierungsbäumchen (treeview) links oben.

b) Aufgerufene Items zeigen

Wenn die rechte Maustaste nicht auf einer **Frage** aktiviert wurde (dann wurden keine anderen Items gebraucht), macht auch der zweite Programmvorschlag Sinn, nämlich die Frage nach den Items, welche von einer Bestimmungsfunktion des angefragten Item aufgerufen wurden. Diese werden dann weiter unten sichtbar, wenn sie nicht schon vorher als Fragen vom Programm gezeigt wurden. Bei einer **Frage** ist die Frage nach den aufgerufenen Items naturgemäß wirkungslos.

c) ExpertItems-Quelltext zeigen

Um eine Übersicht über die Funktion des betreffenden Items zu gewinnen ist es zweckmäßig, sich die Einträge in den Eingabefeldern des Items sich zeigen zu lassen.

d) Kombinationen

Man kann auch Items, die durch die Transparenzfunktion sichtbar wurden, ihrerseits mit der rechten Maustaste anfragen und damit weitere Zusammenhänge untersuchen. Dadurch wird aber der Zusammenhang mit dem ursprünglich angefragten Item verwischt. Um die Übersicht über die Zusammenhänge zu behalten ist es daher zweckmäßig, einzelne dieser Transparenzausgaben wieder zu schließen, was auch dort, wo die Option angeboten wurde, auch als Rückgängig-Machen immer

angeboten wird. Immerhin kann man aber durch Nacheinander-Fragen herausfinden, wodurch sich ein möglicherweise fragwürdiger Wert eines Items ergeben hat.

XVI. Programmieren mit Klassen (Classes)

ExpertItems ist in C++ programmiert. ExpertItem-Codes können in C++ Codes eingefügt werden und umgekehrt.

Die besondere Mächtigkeit von C++ beruht auch in der Möglichkeit **classes** zu verwenden.

In ExpertItems steht an erster Stelle die Frage

IT-Expertenmodus

also ob mit **Klassen** (classes) gerechnet werden soll oder nicht. Wird nämlich nicht mit **Klassen** gerechnet, dann werden alle verwendeten **Klassen** verborgen (bis auf die „Wurzel" TRoot, auf welche sich die Items der ersten Ordnung beziehen) und der sichtbare Code damit ganz wesentlich vereinfacht. Alle bisher dargestellten Programmierungen können auch ohne **Klassen** erfolgen.

Wird die Frage aber bejaht, dann erscheint im Hauptmenü der Typen neben **Knoten** zusätzlich die Option **Knoten verwendet Klassen**.

Sind nämlich in einem Gesetz wie dem Bafög die Regeln der Einkommensberechnung für unterschiedliche Beteiligte gleich, oder müssen Anfangs- und Endvermögen beider Eheleute beim Zugewinnausgleich übereinstimmend berechnet werden, so ist es zweckmäßig, diese in eine **Klasse** auszulagern, welche im **IT-Expertenmodus** vor der normalen Programmierung in der Wurzel, der **TRoot** – damit sie in der folgenden Programmierung auch gefunden wird – einzufügen ist (die Felder **Name der Oberklasse (up-class)** und **Name der Basisklasse** (**base class)** leer gelassen). Die in diese **Klasse** ausgelagerte Berechnung wird in das Hauptrechenwerk mit übernommen, indem eine **node using classes** auf einer weiter oben eingefügten **Klasse** verweist, indem in dem Eingabefeld

Name der Basisklasse

der Name der ausgelagerten Berechnung eingetragen wird.

Diese wird dadurch Teil der Hauptberechnungen, vorausgesetzt dass die Typenbezeichnung für die **Klasse,** meist **Einzelknoten** mit derjenigen dem aufrufenden **Knoten verwendet Klassen** übereinstimmt. Alle Items in dieser **Klasse** lassen sich dadurch von der Hauptebene aus aufrufen. Umgekehrt gilt das nicht, weil einer so aufgerufenen **Klasse** die Information, woher sie aufgerufen wurde, fehlt.

Basisklasse bedeutet aber auch, dass die Items in dem **Knoten verwendet Klassen,** soweit sie mit den Items der **Klasse** übereinstimmen und dies auch durch Bejahung der Frage **in der Basisklasse vorhanden (in dieser Klasse modifiziert)** bestätigt wird, in der aufrufenden **Klasse** bestimmt werden können und dass diese Information in der aufgerufenen **Klasse** benutzt werden kann.

Schließlich kann auch in der ausgelagerten **Klasse** ein Item auf der Hauptebene aufgerufen werden, indem die nicht mit *up.up. ..* adressiert wird, sondern durch Angabe der *root*, der „Wurzel". So lassen sich **alle** items mit Hilfe der **root** absolut adressieren.

Diese Programmiertechniken wurden in WinFam beim Versorgungsausgleich und dem neuen Zugewinnausgleich verwendet. Mit Hilfe von Oberklasse und Basisklasse lassen sich noch mehr Lösungen erreichen.

XVII. Kopieren von Items, Knoten und Klassen

Bereits erstellte Items und auch ganze Knoten und Klassen können innerhalb einer Datei und von Datei zu Datei kopiert werden. Dazu wird das entsprechende Element im Inhaltsfenster angeklickt und das Item/der Knoten/die Klasse mit dem Menüpunkt Bearbeiten/Kopieren (oder Tastenkombination Strg-C) in die Zwischenablage kopiert und anschließend ein Element am Zielort angeklickt und das kopierte Element mit Bearbeiten/Einfügen (bzw. Strg-V) vor dem aktuellen Element eingefügt. Es steht auch das Ausschneiden zur Verfügung (Strg-X). Bei den einleitenden Fragen und Überschriften zu Items, Knoten und Klassen stehen auch im Kontextmenü (über Rechtsklick zugreifbar) die Optionen "Listenelement kopieren", "Listenelement ausschneiden", "Listenelement einfügen", "Listenelement neu einfügen", "Listenelement &löschen" zur Verfügung.

Es können dabei auch Knoten auf Klassen und umgekehrt kopiert werden.

Anhang:

I. Zur Handhabung von ExpertItems

1. Dem Prinzip von ExpertItems entspricht es, wenn man bei Programmieren von dem Ergebniswert ausgeht und dann stufenweise die zu dessen Bestimmung nötigen Items programmiert.

2. Vielfach ist aber der Ergebniswert noch nicht bekannt. Dann muss die gesamte Struktur, die ausprogrammiert werden soll, erst einmal entdeckt werden. Es scheint dann am besten zu sein, wenn man wahllos prüft, welche Zusammenhänge bestehen und diese niederschreibt und dann programmiert. Daran werden sich zwanglos weitere Zusammenhänge anfügen. Viele solche Strukturen wird man ändern müssen, wenn man weiter kommt, und viele Fehler treten auf, weil man bereits Programmiertes nicht an den Gang des weiteren Erkennens der Struktur angepasst hat.

3. Kennt man umgekehrt bereits das Ergebnis und alle Zusammenhänge, dann kann man einen besonders effektiven und schnellen Weg wählen: Alle in anderen Items benötigten Items werden vorweg programmiert. Das hat den Vorteil, dass deren Namen bei der späteren Verwendung in anderen Items sehr einfach aus dem Topic-Balken in die Zwischenablage übernommen und in die Berechnungen eingesetzt werden können, sodass **Schreibfehler vermieden** werden. Man kann zwar auch nach Verfahren 1. die Itemname aus den Eingabefeldern, in denen sie verwendet werden, in die Zwischenablage übernehmen und bei der Anlage des Items verwenden. Das ist aber mühsamer und damit langsamer.

5. Es ist nützlich, immer wieder das bereits erzeugte Programm zu testen und den Testfall für die weitere Verwendung abzuspeichern. Zwar kann in der Internetversion der Testfall nicht gleichzeitig mit dem Programmierfenster eingesehen werden. Man kann sich aber dadurch helfen, dass man unten im Testfall den Button „Export“ betätigt und im *Öffnen*-Menü Öffnen mit Microsoft Office Word wählt. Die gezeigte Ergebnisdatei lässt sich bequem mit dem Inhalt des Programmierfensters vergleichen.

6. Jedes benötigte Item kann vorläufig mit Hilfe einer Frage bestimmt werden, bevor, in einer weiteren Stufe, die Bestimmung durch andere Items programmiert wird.

7. Um die Folge von Eingaben und Ausgaben den Erwartungen des Anwenders anzupassen, kann man die Bestimmung eines Itemwerts – vor seiner effektiven Nutzung - erzwingen, indem man das Item in einem anderen Item unter „Nach Wertbestimmung“ einträgt, z.B. „itemname();“, wenn „itemname“ der Name des vorweg zu bestimmenden Items ist. Durch Eintrag einer Folge von Items kann so eine Reihenfolge der Fragen festgelegt werden.

8. Zur Orientierung bei etwas größeren Programmen dient die **Suchfunktion**. Diese erreicht man bei der Internetfassung über die Suchfunktion des **Internet-Browsers** (während die PC-Version immer oben rechts ein Suchfeld besitzt). Mit Hilfe der rechten Mausteste kann man einen in der Zwischenablage befindlichen Text in das Suchfeld schreiben.

9. Vielfach wird man die Programmierung **vorbereiten** müssen. Wie bei einem Aufsatz wird an den Anfang

- die **Sammlung der Begriffe** gehört, welche im Zusammenhang miteinander stehen. Sodann wird man sich bemühen,

- die **Zusammenhänge der Begriffe** untereinander zu ermitteln und endlich den Begriffen

- ein oder mehrere **Items zuzuordnen** deren Zusammenhang dann das Programm erzeugen kann.

10. Kommentarfunktion:

Der Programmlauf wird weitegehend von Bedingungen gesteuert. Diese bilden die Voraussetzungen für das Ergebnis, lassen aber meist nicht erkennen, welches Ergebnis sie ermöglichen sollen. Deshalb ist es vorteilhaft, für spätere Bearbeitungen den Zweck einer solchen Bedingung als Kommentar zu vermerken. Dazu steht überall die Kommentarfunktion von WinFam zur Verfügung: in der Windowsfassung als Büroklammer in der Funktionsleiste, bei der Internetfassung als Unterfunktion von „Aktion“.

II. Fehlerbehandlung:

Grundlegend für die Fehlerbehandlung bei ExpertItems ist die Frage: welches Item ist fehlerhaft? Ist dieses identifiziert, dann muss seine Arbeitsweise gedanklich genau nachvollzogen werden, um den Fehler beseitigen zu können.

Sechs Fehlerarten sind zu unterscheiden:

1. **Beim Editieren** wird gemeldet, dass **keine Tabulatoren zulässig** seien. Dann enthält ein Eingabefeld einen Tabulator. Das kann geschehen, wenn in ein **Kommentarfeld** ein mit Word vorbearbeiteter Text mit copy-and-paste eingefügt wurde, um das Programm für die Programmpflege zu erläutern. Leider zeigt ExpertItems den Tabulator nicht an, so dass er uU schwer zu finden ist. Es empfiehlt sich dann, mit copy-and-paste den Text noch einmal nach Winword zu lagen und sich mit den entsprechenden Option die Position der Tabulatoren anzeigen zu lassen. Die Korrektur erfolgt dann aber am besten direkt im Kommentarfeld von ExpertItems.

2. **Die Kompilation** scheitert.
   Der Compiler meldet in einem Html-Editor einen Fehler:
   Dann meldet ExpertItems, in welchem Item sich der Fehler befindet.

* Die weitaus häufigste Fehlermeldung dürfte sein: „**xxx unbekannt**“. Dann ist
- entweder vergessen worden, ein Item dieses Namens Item anzulegen, oder
- der Item-Name wurde falsch geschrieben. Schließlich kann auch
- das vorhandene nicht gefunden worden sein, weil es nicht richtig adressiert wurde (es wurde von einer Liste aus aufgerufen und es fehlte „up.up.“ (vgl. VIII.5.:Addressierung in Listen).
- das vom Programm als Item-Name gesuchte Wort ist überhaupt kein Item und wird nur deshalb als ein solches gesucht, weil vergessen wurde, die nötigen **Anführungszeichen** zu setzen.
Schließlich kommt es auch vor, dass man ein Item programmiert und dann vergessen hat. Immer wenn ein Item als nicht vorhanden gemeldet wird, müssen ALLE Stellen gesucht werden, an denen es vorkommt, also: aufgerufen wird!

Wenn bei der Text-Addition ein **Pluszeichen vergessen** wurde, meldet das Programm leider oft, dass eine **Klammer** falsch gesetzt wurde (weil erst bei der Prüfung der Klammern dem Programm der Fehler auffällt).

Mit der **Suchfunktion** (im PC-Programm kleines Eingabefenster rechts oberhalb des Arbeitsblattes, in der Internetfassung zu erreichen mit der Suchfunktion des Browsers, bei vielen Browser über <Strg F>) können die Stellen, an denen sich der **problematische Ausdruck** befindet, gefunden werden. Wenn z.B. versehentlich derselbe Item-Name zweimal verwendet wurde, findet man das Problem nur dadurch, dass man das Vorkommen des Namens in der ganzen Datei sucht.

Andere Syntaxfehler haben meist damit zu tun, dass die Funktionalität von C oder C++ in Anspruch genommen wurde und gegen deren Syntax verstoßen wurde.

In jedem Fall ist das fehlerhafte Item nach Syntaxfehlern zu durchsuchen. Z.B. kann ein Text als Bestimmungsfunktion nur in ein Text-Item eingegeben werden, nicht in ein Geldbetrags-Item, sonst gibt es Konvertierungsprobleme.

Wichtig: es kommt vor, dass das Programm den Fehler erst bemerkt, wenn es das folgende Item bearbeitet. Deshalb muss der Fehler in den **vorhergehenden Items** gesucht werden, wenn er in dem angegebenen nicht gefunden werden kann.

Wenn sich ein angezeigter Fehler nicht finden lässt, muss man andere, die sich finden lassen beseitigen. Bei dem unauffindbaren Fehler handelt es sich dann oft um einen „**Folgefehler**“, also einen solchen, der vom System infolge eines anderen Fehlers irrig angezeigt wird.

Die Fehlermeldung **überladene Funktion** bedeutet, dass versucht wurde, dem Item einen Wert zuzuweisen, der nicht seinem Typ entspricht, also etwa einem TextItem einen Betrag oder umgekehrt einem BetragsItem einen Text. Diese Fehlermeldung wird vom C++-Compiler geliefert.

Zuweilen reklamiert der Compiler, dass Zeichenketten (Texte) **unterbrochen** seien. Dann ist es nützlich bei der beanstandeten Stelle Leerzeilen zu beseitigen.

Sonst sollte auf **Kleinigkeiten** geachtet werden, wie z.B. einen Punkt, der dort nicht hingehört.

3. Das erfolgreich kompilierte Programm liefert **falsche Ergebnisse,** stellt **unnötige Fragen** oder liefert die Fehlermeldung „… **wird zu seiner eigenen Bestimmung benutzt**“.
Für die Auffindung dieser Fehler ist ExpertItems mit der Transparenzfunktion ausgerüstet, welche (auch dem Letztanwender) die Funktionalität des Programms transparent machen soll.

a) Wenn das **Ergebnis falsch ist**, muss der Experte sich folgende Fragen stellen

* Welches Item liefert das falsche Ergebnis?

Diese Frage beantwortet das Programm (in dem Programmlauf mit dem falschen Ergebnis) dann, wenn man mit der Maus auf das falsche Ergebnis geht und mit der rechten Maustaste anklickt (Bei der CD-Version muss man vorher über das Hängemenü „Optionen/Menüpunkte zur Berechnungstransparenz ein/aus“ die Transparenzfunktion einschalten.). Das Programm antwortet mit der Frage, ob das Item xxx gezeigt werden soll und teilt dadurch mit, dass dieses Item den falschen Wert geliefert hat. Die nächste Frage an sich selbst muss lauten:

* Rechnet das Item selbst richtig?

Zu deren Beantwortung muss in dem von der rechten Maustaste gezeigten Menü die Option „Item xxx anzeigen“ gewählt werden. Es zeigt dann drei Fragen:
- Auswertungsgrund zeigen
- Aufgerufene Items zeigen
- ExpertItems Quelltext zeigen

Da nach einem Fehler in der Programmierung des Items gesucht werden soll, muss gewählt werden
- ExpertItems Quelltext zeigen.

Es erscheinen dann alle in dem Item gestellten Fragen mit den zugehörigen Antworten. Wenn diese Prüfung keinen Fehler ergibt, muss der Fehler in einem (oder vielleicht auch mehreren) etwa verwendeten anderen Item(s) liegen, welches einen falschen Wert aufweist. Die Frage an sich selbst muss daher lauten:

*welche Items hat das Item verwendet?

Diese Frage beantwortet das Programm, wenn man von den oben angezeigten Fragen die Frage
- Aufgerufene Items zeigen

bejaht.
Das Programm reagiert damit, dass es die verwendeten Items und ihre Werte anzeigt. (Falls in ExpertItems programmierte Funktionen verwendet wurden, wird auch deren Name angezeigt mit der Option, ihren Quelltext einzusehen.) Es muss nun geprüft werden, welcher der verwendeten Werte nicht korrekt ist. Ist dieser Wert (und das zugehörige Item) gefunden, dann wiederholt sich das Verfahren mit dem neuen Item und der oben angegebenen Frage „Rechnet das Item richtig?“.

Wird dieses Verfahren sorgfältig durchgeführt, muss die Ursache aller falschen Ergebnisse gefunden werden.

b) Wenn das Programm **unnötige Fragen** stellt, dann ist anders zu verfahren:
Die erste Frage an sich selbst lautet wieder:

* Welches Item stellt die unnötige Frage?

Dies Item wird in gleicher Weise gefunden, wie oben bei der Frage „Welches Item liefert das falsche Ergebnis?“ beschrieben. Es wird dann nicht die Ausgabe, sondern die Frage mit der rechten Maustaste angeklickt. Auch hier folgen die Angebote

- Auswertungsgrund zeigen
- Aufgerufene Items zeigen
- ExpertItems Quelltext zeigen

Die unnötige Frage kann zwei Ursachen haben: entweder kann der Wert des Items schon aus dem Wert anderer bereits bestimmter Item erschlossen werden – dann ist das fragende Item falsch programmiert - , oder ein anderes Item hat das fragende Item unnötigerweise aufgerufen. Um die erste Möglichkeit auszuschließen, lautet die erste Frage an sich selbst wieder:

* rechnet das Item richtig?

Wenn der Wert des Items auch ohne die Frage bestimmt werden kann, dann muss dies im Allgemeinen geschehen. Hierfür eignet sich die „Bedingung für ein besonderes Ergebnis“ Das setzt allerdings voraus, dass die Items, aus denen sich der Wert herleitet, bereits bestimmt sind. Dementsprechend muss die Frage „*Bedingung*.is_Set()“ (vgl. IV.4.c) vorangestellt werden, also „*Bedingung*.is_Set() UND *Bedingung*“. Wenn also die Bedingung bereits bekannt ist, kann daraus der Wert der Items in dem „Besonderen Ergebnis“ berechnet und damit die Eingabe gespart werden. Ist das Item korrekt, dann kann das Item auch unnötigerweise aufgerufen worden sein. Deshalb ist zu fragen

* wurde das Item unnötig aufgerufen?

Aus dem Angebot
- Auswertungsgrund zeigen
- Aufgerufene Items zeigen
- ExpertItems Quelltext zeigen

ist dann „Auswertungsgrund zeigen“ zu wählen. Geschieht das, dann zeigt das Programm das Item, welches das fragende Item aufgerufen hat und auch alle Items, die diesen Aufruf mittelbar verursacht haben. Eines dieser Items muss dann einen Fehler enthalten, wenn das Item unnötig aufgerufen wird. Wieder ist für jedes dieser Item die Frage an sich selbst zu stellen

* rechnet das Item richtig?

(weiter s.o.)

c) Fehlermeldung „… **wird zu seiner eigenen Bestimmung benutzt**“
Diese Fehlermeldung soll sog. „Endlosschleifen“ verhindern. Wenn zwei Items sich gegenseitig aufrufen, kommt der Programmlauf zu keinem Ende. Das Programm gibt dann auch an, von welchem Item der unzulässige Aufruf kam. Dieses Item muss dann überprüft werden. Wenn statt eines Items eine Funktion verwendet wurde, suchen. **Hilfreich** ist es hier, wenn man mit der rechten Maustaste in der Fehlermeldung des Items klickt, **über welches** die fehlerhafte Bestimmung erfolgt, und nach Erscheinen der Item-Informationen die Option „Auswertungsgrund zeigen“ wählt. Dann erscheint in dem Übersichtsbäumchen links die ganze Aufrufkette, und man kann auf jeden verdächtigen Punkt zugreifen.

d) Zuletzt mag noch der Fall, dass eine **Frage zu spät erscheint,** erörtert werden. Dieser ist grundsätzlich unproblematisch. Das Item kann nach vorne gezogen werden, indem man in einem anderen Item dieses Item gesondert aufruft. Das kann „Nach Wertbestimmung“ erfolgen, indem dort der *„Itemname();“* eingetragen wird, aber auch in einem anderen Feld (mit Ausnahme

solcher, in welche man Texte ohne Anführungszeichen eintragen kann) durch Komma getrennt, den Aufruf des Items voranstellt, z.B. im Feld Frage: *Itemname(), „Fragetext“*.

4. Kommt man damit nicht zum Ziel, dann kann man sich bestimmte **Zwischenergebnisse** einer laufenden Rechnung auf dem Bildschirm ausgeben lassen. Zu diesem Zweck fügt man in den Quell-Code ausschließlich zum Zweck der Fehlersuche Ausgabefunktionen ein. Dafür ist
   XI_test(<Ausdruck>) und
   XI_test1(<Ausdruck>) vorgesehen.
   <Ausdruck> ist ein beliebiger Rechenausdruck, der auch Items enthalten kann, z.B.
   XI_test(*Bruttolohn-Abzuege*), wenn „Bruttolohn“ und „Abzuege“ Items des Typs „**Geldbetrag**“ sind. Es wird dann der Ausdruck als Text und sein Wert auf dem Bildschirm gemeldet. Beide Funktionen liefern zusätzlich auch noch den ausgegebenen Wert an den Rechenausdruck, in welchem sie erscheinen, sodass sie in größeren Rechenausdrücken verwendet werden können.
   XI_test1() liefert nicht den Itemnamen.

5. Das Programm hängt sich auf
   Das kann geschehen, wenn eine Zählfunktion programmiert wurde, bei der die Bedingung für das Listenende nie erfüllt ist. Dann liegt eine Wiederholung der Berechnung ohne den dazu nötige Abbruchbedingung vor, nach der man dann im programmierten Code suchen muss. Vorher muss der Programmlauf allerdings abgebrochen werden. Dazu nützt die Tastenkombination <strg-alt-entf>, welche die Möglichkeit, einen bestimmten Prozess abzubrechen, eröffnet. Notfalls kann der Rechner heruntergefahren oder einfach abgeschaltet werden.

6. Das Programm stürzt ab
   Sollte nicht passieren, solange nur ExpertItems genutzt wird, andernfalls liegt ein unerkannter Fehler bei ExpertItems vor, den wir zu melden bitten. Wurde in C/C++ programmiert, ist der Quelltext nach Syntaxfehlern durchzusehen, notfalls mit Hilfe eines Fachmanns.

III. Programmierratschläge:

Weiterfragen nur wenn nötig

Bei der Programmierung sollte eine unnötige Belastung des Endanwenders durch Fragen vermieden werden. Hierfür stellt das Programm die Möglichkeit zur Verfügung, Fragen nach einem Ergebnis mit dessen Berechnung so zu kombinieren, dass nur dann berechnet wird, wenn der Anwender das Ergebnis nicht selbst einträgt. Es ist empfehlenswert, in diesem Fall für die eigentliche Berechnung grundsätzlich ein weiteres Item zu verwenden, weil in dessen Bereich eindeutig festliegt, dass kein Ergebnis eingegeben wurde und danach also auch nie gefragt werden muss.

z.B.:

Itemname: Wirksamkeit

Frage: ist der Vertrag wirksam? (ja/nein/unbestimmt)

Rechenausdruck: Wirksamkeit1

Ergebnistext für berechnetes Ergebnis:““

Itemname: Wirksamkeit1

Bedingung für ein besondere Ergebnis: <Itemname1> UND <Itemname2>

Besonderes Ergebnis Ergebnis: false

Ergebnistext: „Das Rechtsgeschäft ist nicht wirksam, weil ….“

…

Frage:““

Rechenausdruck:<Itemname1> UND <Itemname2>UND… ODER <Itemname3> UND…

Ergebnistext: „Das Rechtsgeschäft ist wirksam weil…“

Programmieren: unnötige Meldungen vermeiden

Bei der Programmierung sollten Meldungen vermieden werden, die im konkreten Fall nicht benötigt werden. Hier kann die Programmiererleichterung durch den Programmvorschlag von Meldungen zum Problem werden! Man kann durch Verwendung der **Bedingung für ein besonderes Ergebnis** bei den ausgegebenen Texten differenzieren.

Bedingungen oder neue Items?

Vielfach kann man Arbeit sparen, indem man dem gleichen Item Bedingungen hinzufügt. Das kann aber am Ende unübersichtlich werden. Verwendet man neue Items, dann kann man ihnen passende Namen geben und sie übersichtlich ordnen. Das verbessert das Debugging und Pflegfähigkeit des Programms. Die Bedingungen machen das Programm demgegenüber kompakter. Empfehlenswert ist daher ein Mittelweg: nur wenige Bedingungen, sofern sie nicht sehr gleichartig sind.

IV. Besondere Programmierprobleme

Versuche

In vielen Fällen willen man Berechnungen ausprobieren und muss sie verwerfen und neu rechnen, wenn eine Bedingung nicht erfüllt ist. In diesem Fall kann man eine nummerierte Liste für die einzelnen Versuche anlegen, in welche kein Name eingetragen wird. Dann fragt das Programm die Bedingung für das erste leere Element ab. Hier ist dann einzutragen, dass das vorletzte Element den erfolgreichen Versuch darstellt. Für Das Ergebnis – nämlich das letzte Element der Liste – kann dann mit einem Item des Typs „Listenelemente zählen“ erfasst werden.

Beispiel:

Bei mehreren Bedürftigen, welche mehr oder weniger Eigeneinkommen haben wird im Mangelfall der berücksichtigte Bedarf (und nicht der Unterhalt) gekürzt. Dann scheiden die Berechtigten aus der Verteilung aus, welche den gekürzten Bedarf mit ihrem Eigeneinkommen decken können. Da sie mit ihrem Überschuss nicht zur Bedarfsdeckung der anderen beitragen müssen, scheiden sie aus der Berechnung aus und diese muss wiederholt werden, bis keiner der restlichen Berechtigten mehr mit seinem Einkommen den Bedarf decken kann.

V. Programmierbeispiele:

Beispiele können nur die Arbeitsweise zeigen. Den Nutzen des Programms können sie nicht zeigen, weil sie naturgemäß einfach sein müssen, während erst die Programmierung größerer und komplexerer Zusammenhänge dem Nutzer Zusatzinformationen verschaffen können.

1. Familienrechtliches Minimalprogramm:

Der Anspruch einer nichtehelichen Mutter und ihres Kindes in den ersten drei Jahren berechnet werden.

Logisch-mathematische Vorarbeit:
Es gelten folgende Zusammenhänge:

a) Kindesunterhaltsanspruch:
wenn der volle Kindesunterhalt < Einkommen – 1080: voller Kindesunterhalt
sonst: Einkommen – 1080

b) Einkommen (des Vaters): muss erfragt werden

c) voller Kindesunterhalt = Tabellenunterhalt – 95 (halbes Kindergeld)

d) Tabellenunterhalt
wenn: Einkommen<1500 Tabellenunterhalt=335
wenn: Einkommen <1900 Tabellenunterhalt=352
wenn: Einkommen <2300 Tabellenunterhalt=369
wenn: Einkommen <2700 Tabellenunterhalt=386
wenn: Einkommen <3100 Tabellenunterhalt=402
wenn: Einkommen <3500 Tabellenunterhalt=429
wenn: Einkommen <3900 Tabellenunterhalt=456
wenn: Einkommen <4300 Tabellenunterhalt=483
wenn: Einkommen <4700 Tabellenunterhalt=510
wenn: Einkommen <5100 Tabellenunterhalt=536
wenn: Einkommen nach Einzelfall fragen

d) Unterhaltsanspruch (der Mutter):
wenn Bedürftigkeit< Leistungsfaehigkeit UND Bedürftigkeit < Halbteilung: Bedürftigkeit
wenn Bedürftigkeit<Leistungsfaehigkeit UND Bedürftigkeit > Halbteilung: Halbteilung
sonst: Leistungsfähigkeit

e) Bedürftigkeit: Bedarf – Eigeneinkommen

f) Eigeneinkommen muss erfragt werden.

g) Bedarf:
hypothetisches Einkommen > 880: hypothetisches Einkommen
sonst: 880

h) hypothetisches Einkommen muss erfragt werden

i) Leistungsfähigkeit = Einkommen (des Vaters) – Kindesunterhaltsanspruch – 1200 (Partnerselbstbehalt)

j) Halbteilung = (Einkommen – Kindesunterhaltsanspruch – Eigeneinkommen)*1/2

i) Einkommen (des Vaters) muss erfragt werden

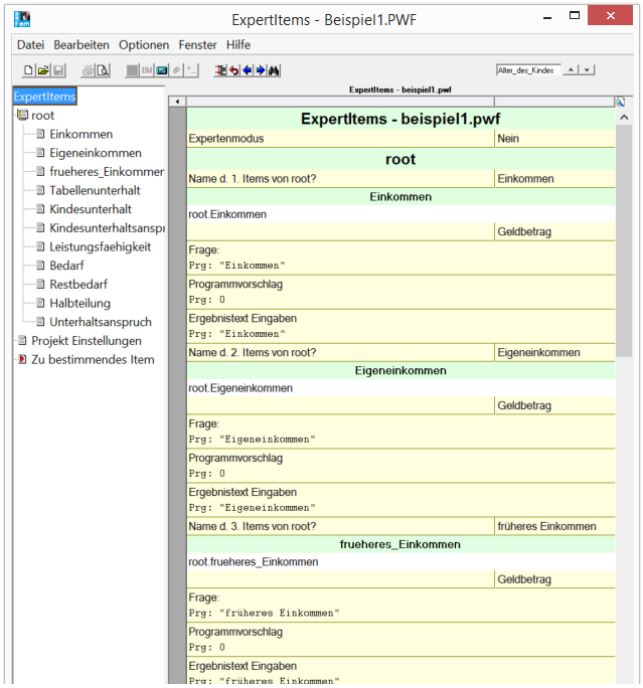
Es werden also 11 Items benötigt, von denen die letzten drei nur aus Fragen bestehen, das zweite auf Frage und Zusammenhang und die übrigen sieben nur aus Zusammenhängen.

Wenn man mit den Frage-Items beginnt, kann man sich das Schreiben bei Verwendung des Items ersparen, indem man die aus der Überschrift herauskopiert. Das erspart Schreibfehler bei der Verwendung eines Items. Wenn man allerdings einfach drauflos programmiert, geht man am besten vom Ergebnis aus – so wie bei Erstellung der obigen Übersicht vorgegangen wurde Man kann nun die Analyse bereits beim Programmieren vollziehen. Dann wird die obige Folge in Items 11 umgesetzt.

Hat man die Zusammenhängen aber bereits wie oben analysiert, dann beginnt man am besten mit der Programmierung der reinen Frage-Items und schreibt dann die Items, deren Wert von diesen bestimmt werden. Weil die anderen Items aber auch voneinander abhängen müssen sie, damit man immer kopieren kann, noch so geordnet werden, dass die unteren Items nur auf die oberen zugreifen. Das wäre etwa folgende Reihenfolge (konstante Beträge in KAPITALIEN):,
a) Einkommen (des Pflichtigen)
b) Eigeneinkommen (des Berechtigten)
c) frueheres_Einkommen des Berechtigten

d) Tabellenunterhalt (aus Einkommen und KINDESALTER)
e) Kindesunterhalt (aus Tabellenunterhalt und KINDERGELD)
f) Kindesunterhaltsanspruch (aus Kindesunterhalt und EIGENBEDARF1):
g) Leistungsfaehigkeit (aus Einkommen, Kindesunterhaltsanspruch und EIGENBEDARF2)
h) Bedarf (aus frueheres_Einkommen und MINDESTBEDARF)
i) Restbedarf (aus Bedarf und Eigeneinkommen
j) Halbteilung (aus Einkommen, Kindesunterhaltsanspruch, Eigeneinkommen, GATTENQUOTE)
k) Unterhaltsanspruch (aus Restbedarf, Leistungsfaehigkeit, Halbteilung)

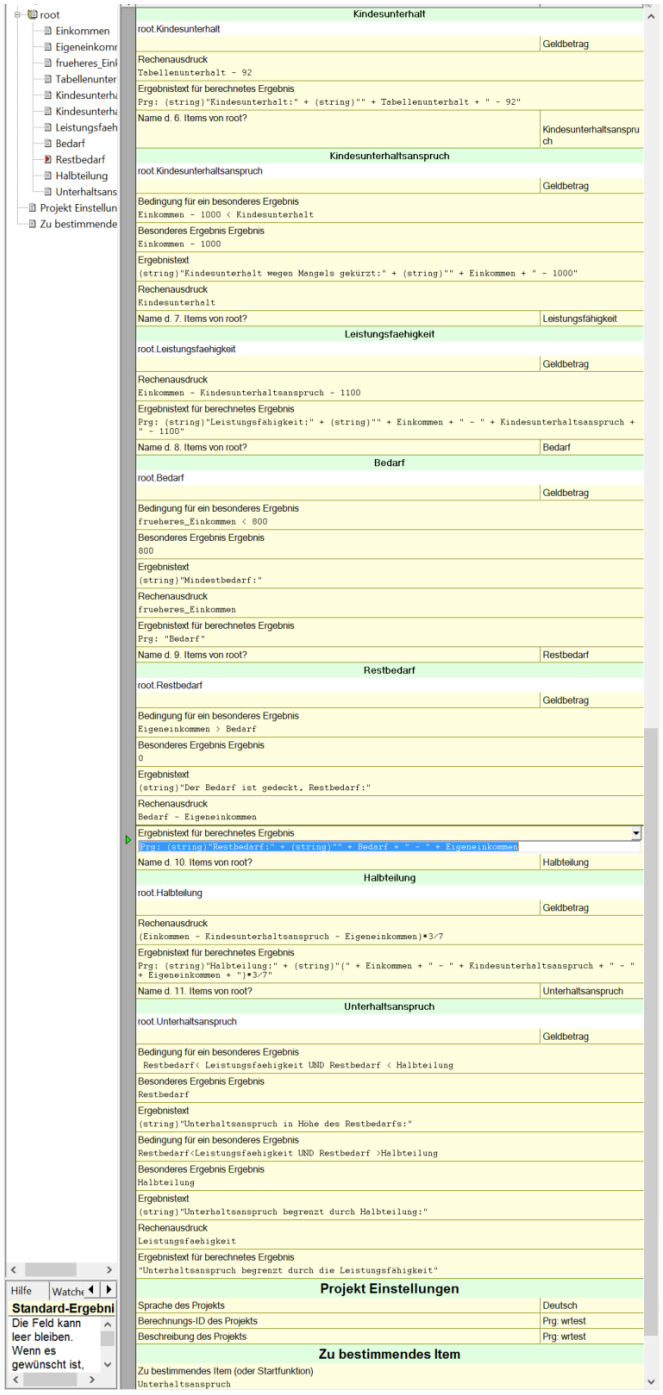
Das ergibt folgendes Programm (wobei die Leerzeilen ausgeblendet wurden)



Es genügte für die drei ersten Items, den Item-Namen einzugeben. Das Programm hat daraus die Frage gemacht und die Ergebnismeldung. Das Item „Tabellenunterhalt nutzt die Bedingungsfunktion. Die Standardausgabe eignet sich, dafür, den Hinweis auf die Einkommensgruppe einzutragen.


ExpertItems - Beispiel1.PWF
Datei Bearbeiten Optionen Fenster Hilfe
Alter_des_Kindes
Tabellenunterhalt
ExpertItems
root
Einkommen
Eigeneinkommen
frueheres_Einkommer
Tabellenunterhalt
Kindesunterhalt
Kindesunterhaltsansp
Leistungsfaehigkeit
Bedarf
Restbedarf
Halbteilung
Unterhaltsanspruch
Projekt Einstellungen
Zu bestimmendes Item
Hilfe
Watcher (DEBUG)
Programmvorgaben:
Um dem
Letztanwender der
Tabellenunterhalt
root Tabellenunterhalt
Geldbetrag
Bedingung für ein besonderes Ergebnis
Einkommen<=1500
Besonderes Ergebnis Ergebnis
317
Ergebnistext
(string)"Tabellenunterhalt Gruppe 1:"
Bedingung für ein besonderes Ergebnis
Einkommen<=1900
Besonderes Ergebnis Ergebnis
333
Ergebnistext
"Tabellenunterhalt Gruppe 2:"
Bedingung für ein besonderes Ergebnis
Einkommen<=2300
Besonderes Ergebnis Ergebnis
349
Ergebnistext
"Tabellenunterhalt Gruppe 3:"
Bedingung für ein besonderes Ergebnis
Einkommen<=2700
Besonderes Ergebnis Ergebnis
365
Ergebnistext
"Tabellenunterhalt Gruppe 4:"
Bedingung für ein besonderes Ergebnis
Einkommen<=3100
Besonderes Ergebnis Ergebnis
381
Ergebnistext
"Tabellenunterhalt Gruppe 5:"
Bedingung für ein besonderes Ergebnis
Einkommen<=3500
Besonderes Ergebnis Ergebnis
406
Ergebnistext
"Tabellenunterhalt Gruppe 6:"
Bedingung für ein besonderes Ergebnis
Einkommen<=3900
Besonderes Ergebnis Ergebnis
432
Ergebnistext
"Tabellenunterhalt Gruppe 7:"
Bedingung für ein besonderes Ergebnis
Einkommen<=4300
Besonderes Ergebnis Ergebnis
457
Ergebnistext
"Tabellenunterhalt Gruppe 8:"
Bedingung für ein besonderes Ergebnis
Einkommen<=4700
Besonderes Ergebnis Ergebnis
482
Ergebnistext
"Tabellenunterhalt Gruppe 9:"
Bedingung für ein besonderes Ergebnis
Einkommen<=5100
Besonderes Ergebnis Ergebnis
508
Ergebnistext
"Tabellenunterhalt Gruppe 10:"
Frage
"Kindesbedarf nach den Umständen des Falles"
Programmvorschlag
508
Die Antwort ist falsch, wenn ...
Einkommen>5100 UND Tabellenunterhalt <510
Meldung bei Fehleingabe
"nicht weniger als 510!"
Ergebnistext Eingaben
"Kindesbedarf nach den Umständen des Falles"

Die folgenden Items sind nach Ausblenden der Leerzeilen recht übersichtlich:



Die Rechenausdrücke, welche dem Anwender erklären, wie das berechnete Ergebnis zustande kommt, werden vom Programm vorgeschlagen (daher “Prg:“)

Man darf nicht vergessen, am Ende das zu **bestimmende Item**, hier den Unterhaltsanspruch, einzutragen.

2. Zivilrechtliche Minimalprogramm:

Abschluss eines Vertrags (nur Logik, keine numerisches Rechenwerk)

Logische Vorarbeit:

a). Vertrag ist wirksam, wenn

- gültiges Angebot

- gültige Annahme

- kein Wirksamkeitshindernis

vorliegen

b) gültiges Angebot liegt vor, wenn

- Antragender

  - geschäftsfähig ist

  oder sein gesetzlicher Vertreter gehandelt hat

- oder als vermindert Geschäftsfähiger gehandelt hat und der gesetzliche Vertreter genehmigt hat

- die Erklärung nicht wirksam angefochten wurde

c) gültige Annahme liegt vor, wenn

- Annehmender

  - geschäftsfähig ist

  oder sein gesetzlicher Vertreter gehandelt hat

- oder als vermindert Geschäftsfähiger gehandelt hat und der gesetzliche Vertreter genehmigt hat

- die Erklärung nicht wirksam angefochten wurde

- und die Annahme innerhalb der Annahmefrist erklärt wurde

d) die Annahme wurde innerhalb der Annahmefrist erklärt, wenn

- unter Anwesenden oder telefonisch angeboten und sofort angenommen wurde

- oder eine Annahmefrist gesetzt war und die Annahmefrist eingehalten wurde

- die Annahme hätte rechtzeitig zugehen müssen und der Antragende die Verspätung nicht unverzüglich gerügt hat.

e) Kein Wirksamkeitshindernis liegt vor wenn

- Vertrag nicht gegen ein gesetzliches Verbot verstößt,

- und Vertrag nicht sittenwidrig und nicht wucherisch ist

- kein Dissens vorliegt

- keine Erklärung wegen Täuschung oder Drohung erfolgreich angefochten wurde

- keine Erklärung wegen Irrtums erfolgreich angefochten wurde

f) wegen Täuschung oder Drohung wurde erfolgreich angefochten wenn

- die Erklärung durch Täuschung oder Drohung verursacht wurde und

- binnen einen Jahres deswegen die Anfechtung erklärt wurde.

g) wegen Irrtum wurde erfolgreich angefochten, wenn

- die Erklärung auf einem Irrtum über den Erklärungsinhalt oder eine verkehrswesentliche Eigenschaft einer beteiligten Person oder Sache verursacht wurde und

- die deshalb unverzüglich die Anfechtung erklärt wurde.

Hier zeigt sich, dass für Angebot und Annahme vielfach dieselben Frage zu klären sind. Hier kann die Ordnung der Items mit Hilfe einer Namensliste (s.o. X.1.) helfen. Für alle Items, die doppelt vorkommen, wird eine Namens-Liste angelegt. Die Liste soll den Namen des Oberbegriffs tragen, heißt also „Erklaerung“, die Elemente dieser Namens-Liste werden durch Namen bezeichnet und zwar durch die beiden Namen „Angebot“ und „Annahme“. Die Liste hat also nur zwei Elemente. Zu jedem Element können aber als UnterItems der Liste beliebig viele Unter-Items gehören (durch eine Punkt „.“ Von dem Elementnamen getrennt). Die Wirksamkeit des Angebots kann dann geschrieben werden: Erklaerung[„Angebot“].wirksam. Zur Grammatik: „Erklaerung“ und „wirksam“ sind Items, werden also ohne Anführungszeichen geschrieben, während „Angebot“ der Name eines Elements der Namensliste „Erklaerung“ und deshalb mit Anführungszeichen geschrieben werden muss.

Damit genügen ca. 25 Ja/Nein-Items, den Zusammenhang darzustellen. Da Geschäftsfähigkeit und Vertreten bei beiden Parteien zu prüfen sind, sind gewisse Vereinfachungen möglich, s.u.:

Expertitems - beispiel2.pwf	
Expertenmodus	Nein
root	
Name d. 1. Items von root?	Der Vertrag ist wirksam.
Der_Vertrag_ist_wirksam	
root.Der_Vertrag_ist_wirksam	
	Ja / Nein
Frage: "Der Vertrag ist wirksam."	
Rechenausdruck gueltig["Anbieter"].gilt UND gueltig["Annehmer"].gilt UND uebereinstimmende_Willenserklaerungen UND NICHT Wirksamkeitshindernis	
Ausgabe für "ja" "Der Vertrag ist wirksam."	
Ausgabe für "nein" "Der Vertrag ist nicht wirksam."	
Name d. 2. Items von root?	gueltig
gueltig	
root.gueltig	
	Knoten
	Namensliste
Namen in der Liste können ersetzt werden	Nein
[]	
	Einzelknoten
Name d. 1. Items von root.gueltig[]?	gilt
gilt	
root.gueltig[].gilt	
	Ja / Nein
Überschrift vor der Bestimmung des Itemwerts	XI_name
Bedingung für ein besonderes Ergebnis Willensmaengel	
Besonderes Ergebnis Ergebnis false	
Bedingung für ein besonderes Ergebnis geschaeftsfaehig UND NICHT Willensmaengel	
Besonderes Ergebnis Ergebnis true	
Bedingung für ein besonderes Ergebnis gesetzlicher_Vertreter_hat_gehandelt UND NICHT Willensmaengel	
Besonderes Ergebnis Ergebnis true	
Bedingung für ein besonderes Ergebnis beschraenkt_geschaeftsfaehig UND gesetzlicher_Vertreter_hat_genehmigt UND NICHT Willensmaengel	
Besonderes Ergebnis Ergebnis true	
Rechenausdruck false	
Ausgabe für "ja" "Die Erklärung des "+XI_name+"s ist gültig."	
Ausgabe für "nein" "Die Erklärung des "+XI_name+"s ist nicht gültig."	
Name d. 2. Items von root.gueltig[]?	gesetzlicher_Vertreter_hat_gehandelt
gesetzlicher_Vertreter_hat_gehandelt	
root.gueltig[].gesetzlicher_Vertreter_hat_gehandelt	
	Ja / Nein
Frage: "Der gesetzliche Vertreter hat gehandelt?"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der gesetzliche Vertreter hat gehandelt."	
Name d. 3. Items von root.gueltig[]?	geschaeftsfaehig
geschaeftsfaehig	
root.gueltig[].geschaeftsfaehig	
	Ja / Nein
Frage: XI_name + " ist geschäftsfähig"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der "+XI_name + " ist geschäftsfähig."	
Ausgabe für "nein" "Der "+XI_name + " ist nicht vollgeschäftsfähig."	
Name d. 4. Items von root.gueltig[]?	beschränkt geschäftsfähig
beschraenkt_geschaeftsfaehig	
root.gueltig[].beschraenkt_geschaeftsfaehig	
	Ja / Nein
Frage: "Der "+XI_name +" ist beschränkt geschäftsfähig"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der "+XI_name+ " ist beschränkt geschäftsfähig."	
Ausgabe für "nein" "Der "+XI_name+ " ist auch nicht beschränkt geschäftsfähig."	
Name d. 5. Items von root.gueltig[]?	gesetzlicher Vertreter hat genehmigt
gesetzlicher_Vertreter_hat_genehmigt	
root.gueltig[].gesetzlicher_Vertreter_hat_genehmigt	
	Ja / Nein
Frage: "gesetzlicher Vertreter hat genehmigt"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der gesetzliche Vertreter hat genehmigt."	
Ausgabe für "nein" "Der gesetzliche Vertreter hat nicht genehmigt."	
Name d. 6. Items von root.gueltig[]?	Willensmängel
Willensmaengel	
root.gueltig[].Willensmaengel	
	Ja / Nein
Frage: "bestehen Willensmängel bei der Erklärung des "+XI_name+"s."	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Rechenausdruck NICHT Geschaeftswille ODER Taeuschungsanfechtung ODER Irrtumsanfechtung	
Ausgabe für "ja" "Es bestehen Willensmängel bei der Erklärung des "+XI_name+"s."	
Ausgabe für "nein" "Es bestehen keine Willensmängel bei der Erklärung des "+XI_name+"s."	
Name d. 7. Items von root.gueltig[]?	Geschäftswille
Geschaeftswille	
root.gueltig[].Geschaeftswille	
	Ja / Nein
Frage: "bei dem " +XI_name+" (bzw. seinem Vertreter) war der Geschäftswille vorhanden"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Bei "+ XI_name+" (bzw. seinem Vertreter) war der Geschäftswille vorhanden."	
Ausgabe für "nein" "Bei "+ XI_name+" (bzw. seinem Vertreter) war kein Geschäftswille vorhanden."	
Name d. 8. Items von root.gueltig[]?	Taeuschungsanfechtung
Taeuschungsanfechtung	
root.gueltig[].Taeuschungsanfechtung	
	Ja / Nein
Rechenausdruck Taeuschung UND Anfechtung_binnen_Jahresfrist	
Ausgabe für "ja" "Die Erklärung wurde wegen Täuschung erfolgreich angefochten."	
Name d. 9. Items von root.gueltig[]?	Täuschung

Taeuschung	
root.guellig[].Taeuschung	
	Ja / Nein
Frage: "Wurde der "+XI_name+" (oder Vertreter) zu der Erklärung durch Täuschung bewegt"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der "+XI_name+" (oder Vertreter) wurde zu der Erklärung durch Täuschung bewegt."	
Name d. 10. Items von root.guellig[]?	Anfechtung_binnen_Jahresfrist
Anfechtung_binnen_Jahresfrist	
root.guellig[].Anfechtung_binnen_Jahresfrist	
	Ja / Nein
Frage: "Die Erklärung wurde binnen Jahresfrist wegen Täuschung angefochten"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Die Erklärung wurde binnen Jahresfrist wegen Täuschung angefochten."	
Ausgabe für "nein" "Die Erklärung wurde nicht binnen Jahresfrist wegen Täuschung angefochten."	
Name d. 11. Items von root.guellig[]?	Irtumsanfechtung
Irrtumsanfechtung	
root.guellig[].Irrtumsanfechtung	
	Ja / Nein
Rechenausdruck Irrtum UND unverzuegliche_Anfechtung	
Name d. 12. Items von root.guellig[]?	Irrtum
Irrtum	
root.guellig[].Irrtum	
	Ja / Nein
Frage: "Wurde der "+XI_name+" (oder Vertreter) zu der Erklärung durch einen Irrtum über den Inhalt der Erklärung oder eine verkehrswesentliche Eigenschaft der Person oder Sache bewegt?"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der "+XI_name+" (oder Vertreter) wurde zu der Erklärung durch einen Irrtum über den Inhalt der Erklärung oder eine verkehrswesentliche Eigenschaft der Person oder Sache bewegt."	
Name d. 13. Items von root.guellig[]?	unverzuegliche_Anfechtung
unverzuegliche_Anfechtung	
root.guellig[].unverzuegliche_Anfechtung	
	Ja / Nein
Frage: "Die Erklärung wurde unverzüglich wegen Irrtuas angefochten"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Die Erklärung wurde unverzüglich wegen Irrtuas angefochten."	
Ausgabe für "nein" "Die Erklärung wurde nicht unverzüglich wegen Irrtuas angefochten."	
Name d. 3. Items von root?	uebereinstimmende_Willenserklaerungen
uebereinstimmende_Willenserklaerungen	
root.uebereinstimmende_Willenserklaerungen	
	Ja / Nein
Überschrift vor der Bestimmung des Itemwerts	"Übereinstimmende Willenserklärungen"
Frage: "übereinstiaende Willenserklärungen"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Rechenausdruck Angebot_zugegangen UND Annahme_fristgerecht	
Ausgabe für "ja" "Übereinstiaende Willenserklärungen liegen vor."	
Ausgabe für "nein" "Übereinstiaende Willenserklärungen liegen nicht vor."	
Name d. 4. Items von root?	Wirksamkeitshindernis
Wirksamkeitshindernis	
root.Wirksamkeitshindernis	
	Ja / Nein
Überschrift vor der Bestimmung des Itemwerts	"Wirksamkeitshindernisse"
Frage: "Wirksamkeitshindernis"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Rechenausdruck gesetzliches_Verbot ODER sittenwidrig ODER wucherisch	
Ausgabe für "nein" "Es bestehen keine Wirksamkeitshindernisse."	
Name d. 5. Items von root?	gesetzliches Verbot
gesetzliches_Verbot	
root.gesetzliches_Verbot	
	Ja / Nein
Frage: "Verstößt der Vertrag gegen ein gesetzliches Verbot"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der Vertrag verstößt gegen ein gesetzliches Verbot."	
Name d. 6. Items von root?	sittenwidrig
sittenwidrig	
root.sittenwidrig	
	Ja / Nein
Frage: "Der Vertrag ist sittenwidrig"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der Vertrag ist sittenwidrig."	
Name d. 7. Items von root?	wucherisch
wucherisch	
root.wucherisch	
	Ja / Nein
Frage: "Der Vertrag ist wucherisch"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Der Vertrag ist wucherisch."	
Name d. 8. Items von root?	Angebot_zugegangen
Angebot_zugegangen	
root.Angebot_zugegangen	
	Ja / Nein
Frage: "Ist das Angebot zugegangen"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Das Angebot ist zugegangen."	
Ausgabe für "nein" "Das Angebot ist nicht zugegangen."	
Name d. 9. Items von root?	Annahme fristgerecht
Annahme_fristgerecht	
root.Annahme_fristgerecht	
	Ja / Nein
Frage: "Annahae wurde fristgerecht erklärt"	
Programmvorschlag (ja/nein/unbestimmt)	Nein
Ausgabe für "ja" "Die Annahae wurde fristgerecht erklärt."	
Ausgabe für "nein" "Die Annahae wurde nicht fristgerecht erklärt."	
Projekt Einstellungen	
Sprache des Projekts	Deutsch
Berechnungs-ID des Projekts	Prg: wrtest
Beschreibung des Projekts	Prg: wrtest
Zu bestimmendes Item	
Zu bestimmendes Item (oder Startfunktion) Der_Vertrag_ist_wirksam	
Name der Erzeugungskonfiguration	makewr-wrexpert.ini
Programmcode erzeugen (ohne Compilation)	
Programmcode erzeugen, compilieren und Programm starten	
Ende	

Index

. Bedingung für die Ausgabe 9

. Text für ein berechnetes Ergebnis 9

„New_line()“ 21

Add_string 21

Additionsketten 21

Additionskolonnen 37

Adressierung der Unter-Items 33

Adressierung in einem Unter-Item 34

Adressierung in Listen 32

Antwort ist falsch, wenn 9

Arbeitsblatt 3

Aufzählungen 10, 18

Ausblenden von Leereingaben 27

Bedingung für die Zählung des Elements mit dem Index i 35

Bedingung für ein besonderes Ergebnis 8, 15

Besondere Überschrift für das Übersichtsbäumchen 7

Besonderes Ergebnis 8, 15

Bestimmungsfunktion 3, 5, 12

Bildschirmausgabe 22

boolesche Items 10

Cu2string 21

Currency_output 20

D2string 21

Default_in_vector_since 40

Difference_months 26

Double_output 20

Druckausgabe 22

Einfache Typen 5

Einfügen, Löschen und Verschieben eines Itemnamens 4

Eingabefelder 7

Eingreifen in den automatischen Programmablauf 17

Einzelknoten 33

Empty_input_list_end 35

Empty_line_if_required() 22

ergänzendes Programmieren 17

Ergebnistext 8

ersetzbar 31

Expertenmodus 3, 45

ExpertItems im Internet 46

Fehlerbehandlung 47

First_day_in_month() 26

First_day_next_month() 26

Frage 8

Programmvorschlag 13

Funktionen 40

Ganzzahl 22

Geldbetrag 22

Get_calculation() 23

Get_embraced_calculation(), 23

Get_julian 26

Gruppierungsknoten 28

Heute 26

Hilfefeld 3

Hilfetext 8, 11

Int2string 21

***is_Set()* 15**

Item

Einfache Typen 5

Itemtypen mit einer jeweils ganz speziellen Funktionalität 6

komplexe Typen 5
Löschen, Einfügen und Verschieben eines Itemnamens 4
Namen 4
Rechenausdruck 13
Strukturitemtypen 6
Typ 5
unechte Itemtypen 6
Item direkt eingegeben 15
Items 3
Itemtyp 7
Ja/Nein-Item
Standardergebnistext 16
Ja/Nein-Items 10
Klassen (Classes) 44
Knoten 29
Kolonne ausgeben 37
Kommazahl 22
komplexe Typen 5
Kopieren von Items, Knoten und Klassen 46
Last_day_in_month 26
Last_day_in_month() 26
Last_day_previous_month 26
Leereingaben 27
Listen und Einzelknoten 29
Listenelemente addieren 35
Listenelemente zählen 35
Mehrspaltig rechnen (Vektoren) 39
mehrstufige Listen 32
mehrzeilig 19
Meldung bei Fehleingabe 9
Menüeingabe 18
Menüeingaben 10
Minus_days 26
Minus_months 26
Nach Wertbestimmung 10
Namen des Elements 31
Namensliste 29
Next_input_store_calculations 23
Next_input_store_calculations(). 23
Next_output_print_only() 22
Number_of_workdays 26
Nummerierte Liste 30
Nutzerentscheidung 14
Nutzerentscheidung verlangen 9
Orientierungsbäumchen 3
Percentage_output 20
Plus_days 26
Plus_months 26
Pluszeichen vergessen 48
Programmvorschlag 9
Frage 13
Prozentsatz 22
quantitatives Item
Standard-Ergebnistext 24
Rechenausdruck 9, 14
Item 13
root 3
Round(x,y) 24
Round_down(x,y) 24
Round_up(x,y) 24
Runden 24
Show_string 22
Sonderzeichen 33

Standard-Ergebnistext

Ja/Nein-Item 16

quantitatives Item 24

String_output 20

Strukturitemtypen 6

Tabellen 20

Table_cu2string 21

Table_output 20

Text-Items 18

this_list[i] 31

Today() 26

Topic_Soutp 7

Topic-Ebene 11

Transparenzfunktionen 42

treeview 3

Typ 5

Überschrift des Hilfetextes 7

Überschrift nach Bestimmung des Item-Werts 9

Überschrift nach der Bestimmung des Itemwerts 16

Überschrift vor der Bestimmung des Itemswerts 10

Überschrift vor der Bestimmung des Itemwerts 7

Umsetzen von Items 28

unechte Itemtypen 6

up. 33

upto(Name des Knotens) 33

***Value_from_input()* 15**

Variablen 40

Verwendung der Listen 34

Weitere Verwendung der Listen *Siehe* Listen

Wertbestimmung 3

wird zu seiner eigenen Bestimmung benutzt 50

XI_GetElementName 33

XI_name 33

XI_number 33

Zeilenumbruch 3, 21

Zugeordnete Liste 32