Formeln für Formulare

Die in dieser Dokumentation verwendeten Beispiel-Variablen stammen hauptsächlich aus *tibros*-PP und *tibros*-BB. Die erläuterten Formeln und Funktionen sind selbstverständlich auch auf andere tibros-Programme übertragbar.

Zur Erstellung bestimmter Formeln kann es nützlich sein, in WinWord die Formatierungszeichen einblenden zu lassen. Das geht am schnellsten über den Button  in der Standardsymbolleiste.

Die in dieser Dokumentation erläuterten Funktionen und Formeln beziehen sich auf ältere Word-Versionen, die noch das .doc-Format verwenden. Mit der Einführung der neuen Word-Versionen, die das docx-Format verwenden, hat sich gezeigt, dass in den neuen Versionen einige bisher gebräuchliche Lösungen nicht mehr funktionieren. Soweit dies programmtechnisch in tibros gelöst werden konnte, wurde dies entsprechend angepasst. In einigen Fällen muss jedoch die bisher verwendete Formel im Formular angepasst werden. Diese Sonderlösungen werden am Ende des Dokuments aufgelistet.

Einfügen von Variablen

Eine Variable kann genutzt werden, indem vor die gezeigte Variable "M->" gestellt wird und der Ausdruck in die Klammern « » gesetzt wird.

Beispiel:	Name der Variable	PP_TT[2,1,4]
	Nutzung im Formular	«M->PP_TT[2,1,4]»

Die Klammern erzeugt man mit der Tastenkombination ALT+174 bzw. ALT+175, wobei die Ziffern über den Nummernblock der Tastatur zu tippen sind.

Bei der Verwendung von Variablen ist darauf zu achten, dass diese korrekt in den normalen Text integriert werden, also dass Leertasten und Satzzeichen korrekt verwendet werden. Das ist z. B. bei der Briefanrede relevant:

Beispiel:

Falsch:
Sehr geehrte «PP_TT[9,1,23,1,2,1,16]»

Mögliche Ergebnisse:
Sehr geehrte r Herr Müller
Sehr geehrte Frau Maier
Sehr geehrte Damen und Herren

Richtig:
Sehr geehrte«PP_TT[9,1,23,1,2,1,16]» ,

Mögliche Ergebnisse:
Sehr geehrter Herr Müller,
Sehr geehrte Frau Maier,
Sehr geehrte Damen und Herren,

Die Programmvariablen werden von tibros als Zeichenformat an Word übergeben, daher kann es erforderlich sein, sie zu bestimmten Zwecken in ein anderes Format umzuwandeln. Für die Umwandlung von Feldinhalten gibt es spezielle Funktionen.

«Ctod()»

Die Formulierung ctod() wandelt die Zeichenfolge aus tibros in ein Datumsformat um. Ctod bedeutet „Character to Date“, also ins Deutsche übersetzt „Zeichen zu Datum“. Ohne diese Umwandlung wäre es nicht möglich, die Zeichenfolge aus tibros in verschiedene Datumsdarstellungen zu formatieren. Näheres zu Datumsdarstellungen finden Sie auch unter „Datumsfunktionen“.

«Val()»

Die Funktion Val() wandelt das Zeichenformat aus tibros in ein numerisches Format um, sofern die Variable Zahlen enthält. Dies ist immer dann erforderlich, wenn mit dem Variableninhalt eine mathematische Operation durchgeführt werden soll. Dies kann z. B. ein Größenvergleich sein, wenn abgefragt werden soll, ob die Punkte im Feld XY unter 50 Punkte liegen.

Bei Vergleichen mit numerischen Formaten darf die zu vergleichende Zahl nicht in Hochkommas gesetzt werden!

Beispiele hierzu finden Sie unter iif-Abfragen.

«Str()»

Die Funktion str() wandelt den Inhalt der Klammer zurück in ein Zeichenformat, stellt also das Gegenteil von val() dar.

«substr()»

Die Funktion substr() ermöglicht es, nur einen Teilbereich einer Variable abzufragen. Dies kann z. B. dann erforderlich sein, wenn man abfragen möchte, ob der Prüfling die Prüfung zum ersten Mal ablegt oder ob er evtl. an einer Wiederholungsprüfung teilnimmt. Da das Feld Prüfungsart verschiedene zweistellige Schlüssel enthalten kann (z. B. 10, 11, 12, 20, 21, 22, 90, 91, 92….), wäre es sehr aufwendig, mit der Abfrage auf alle potentiell möglichen Varianten einzugehen. Bezieht man sich jedoch nur auf die zweite Stelle des Inhalts, reduziert das die Möglichkeiten auf die Inhalte 0, 1 oder 2.

Innerhalb der Funktion substr() muss man angeben, auf welche Variable man sich bezieht, welche Stelle der Variable man abfragen möchte und auf wie viele Zeichen des Inhalts man sich beziehen möchte:

«Substr(Variablenname,Beginn-Stelle,Länge)»
Beispiel: «substr(M->PP_TT[4],2,1)»

In obigem Beispiel wäre das also die Variable M->PP_TT[4]. Innerhalb dieser Variable soll die 2. Stelle (also z. B. die 1 von 91) abgefragt werden. Da man sich nur auf 1 Zeichen beziehen möchte, muss man als Länge 1 angeben:

Weitere Beispiele hierzu finden Sie unter „iif-Abfragen“

«Alltrim()», «Ltrim()», «Rtrim()»

Der Befehl alltrim() schneidet alle Leerzeichen am Anfang und am Ende der Variablen ab. In der Klammer wird die Variable angegeben, bei der die Leerzeichen entfernt werden sollen.

Beispiele hierzu finden Sie unter „Leerstellen unterdrücken“.

«Ltrim()» und «Rtrim()» entfernen jeweils nur die Leerzeichen am Beginn (also in Leserichtung „**l**inks“) oder am Ende (also in Leserichtung „**r**echts“) der Variable.

«padl()», «padr()»

Mit dieser Funktion kann man den Inhalt einer Variable auf eine bestimmte Stellenanzahl auffüllen:»

«Padl(Variablenname,Anzahl,Füllzeichen)»
«Padr(Variablenname,Anzahl,Füllzeichen)»

Die Funktion padl ergänzt den Variableninhalt am Beginn (also in Leserichtung „**l**inks“), padr am Ende (also in Leserichtung „**r**echts“) der Variable.

Die Zahl, die bei „Anzahl“ eingegeben wird, repräsentiert die Anzahl Stellen, die die Variable nach der Auffüllung haben soll. Als Füllzeichen kann jedes beliebige Zeichen sein, z. B. eine Leertaste oder eine 0. Wichtig dabei ist, dass das Füllzeichen, wenn es ein Textformat darstellt, in Hochkommas gesetzt werden muss.

Datumsfunktionen:

Datum einfügen:

«FormatDateTime()»

Hier muss die Klammer mit einer entsprechenden Definition gefüllt werden. Es muss angegeben werden, was als Datum ausgegeben werden soll und wie das Datum dargestellt werden soll. Für die Darstellung kann eines der Formate aus der Tabelle unten verwendet werden.

Wie das Datum dargestellt werden soll, wird in der Datumsfunktion als erstes angegeben:

«FormatDateTime('dd. MMMM yyyy', Datumswert)»

oder für das Datum im englischen Format:
«FormatDateTime_e('D MMMM YYYY', ctod(M-> PP_TT[9,1,24,1,7]))»

oder für das Datum im französischen Format:
«FormatDateTime_f('D MMMM YYYY', ctod(M-> PP_TT[9,1,24,1,7]))»

Wichtig ist, dass die Formulierung der Datumsdarstellung in Hochkommas (nicht in die Anführungszeichen „ und “ !) gesetzt wird, danach kommt ein Komma. Nach dem Komma wird angegeben, was als Datum ausgegeben wird. Der Inhalt des Datums kann entweder das aktuelle Tagesdatum sein oder aus einer Programmvariable kommen (z. B. Prüfungsdatum).

Gängige Datumsformate

Formulierung	Ergebnis
dd.MM.yy	01.11.07
dd.MM.yyyy	01.11.2007
dddd, d. MMMM yyyy	Mittwoch, 1. November 2007
d. MMMM yyyy	1. November 2007
dd. MMM. yyyy	01. Nov. 2007
yyyy-MM-dd	2007-11-01

Einfügen aktuelles Tagesdatum:

«FormatDateTime('dd. MMMM yyyy',Date())»

Die Formulierung Date() erzeugt das Tagesdatum aus der Systemzeit des PCs. Wird ein falsches Datum ausgegeben, muss also überprüft werden, ob die Datumseinstellungen von Windows korrekt sind.

Datum aus einer Programmvariable:

Die Programmvariable wird von tibros als Zeichenfolge an Word übergeben, daher muss sie in ein Datum umgewandelt werden. Dies geschieht durch die Formulierung ctod(). Ctod bedeutet „Character to Date“, also ins Deutsche übersetzt „Zeichen zu Datum“.

Beispiel:

Prüfungsdatum: «FormatDateTime('dddd, dd. MMMM yyyy', ctod(M->PP_TT[3,1,2,1,12]))»

Ergebnis:
Prüfungsdatum: Montag, 15. Oktober 2007

Rechnen mit einem Datum aus einer Programmvariable:

Es ist auch möglich, ein Datum unter Verwendung einer Programmvariable zu errechnen.

Beispiel:

Eine Rückantwort wird spätestens 10 Tage vor dem Prüfungsdatum aus dem obigen Beispiel erwartet:

Rückantwort bis: «FormatDateTime('dddd, dd. MMMM yyyy', ctod(M->PP_TT[3,1,2,1,12])-10)»

Ergebnis:
Rückantwort bis: Freitag, 05. Oktober 2007

Das Prüfungsdatum, das in der Variable ctod(M->PP_TT[3,1,2,1,12]) enthalten ist, ist der 15. Oktober. Von diesem werden 10 Tage abgezogen: ctod(M->PP_TT[3,1,2,1,12])-10, also = 5. Oktober.

iif-Abfragen:

Siehe auch "Umstellungsprobleme beim Format docx"!

iif-Abfragen funktionieren ähnlich wie die in Word-Serienbriefen verwendeten Wenn-Dann-Sonst-Abfragen.

Die Grundform der Abfrage ist:

«iif(,,,)»

Diese Grundform muss ergänzt werden durch die Definition der Bedingung und dem, was dann passieren soll:

«iif(Wenn-Bedingung, Dann, Sonst)»

Der Dann- und Sonst-Fall kann sowohl die Ausgabe einer Variable darstellen als auch die Ausgabe von Text. Bei allen Textangaben ist zu beachten, dass diese am Beginn und am Ende jeweils mit Hochkomma gekennzeichnet werden müssen. Nicht verwendet werden dürfen die typographischen Anführungszeichen wie z.B. „und“.

Im Unterschied zur normalen Ausgabe von Variablen werden die Variablen innerhalb der iif()-Funktion nicht noch zusätzlich in die Klammern « und » gesetzt. Dies gilt ebenso für Funktionen wie z. B. val().

Es ist auch darauf zu achten, dass die Wenn-Bedingung, der Dann-Fall und der Sonst-Fall jeweils durch ein Komma getrennt sein müssen. Dies trifft auch dann zu, wenn es keinen Sonst-Fall gibt. Gibt es keinen Sonst-Fall, müssen nach dem Komma noch zusätzlich zwei Hochkommas gesetzt werden. Zwischen den beiden Hochkommas darf in diesem Fall keine Leertaste gemacht werden!

Außerdem ist darauf zu achten, dass der Datentyp im Dann-Fall der gleiche ist wie der im Sonst-Fall. Ist also das Ergebnis im Dann-Fall eine Zahl dann muss das Ergebnis im Sonst-Fall auch eine Zahl sein.

Wenn-Bedingung

Die Wenn-Bedingung wird definiert, indem die Variable angegeben wird und eine Bedingung, was in Bezug auf die Variable zutreffen soll:

M->PP_TT[9,1,7]='J'

Auch hier muss die Textangabe in der Bedingung durch Hochkommas gekennzeichnet sein.

Abweichend hiervon muss vorgegangen werden, wenn ein Zahlenvergleich vorgenommen werden soll, beispielsweise wenn geprüft werden soll, ob ein Prüfungsergebnis unter 50 Punkte liegt. In diesem Fall muss die Variable, die das Prüfungsergebnis enthält, mit val() in ein Zahlenformat umgewandelt werden, die zu vergleichende Zahl (50 Punkte) darf dann nicht in Hochkommas gesetzt werden.

Soll nach einem leeren Feld gefragt werden, lautet die Definition:

M->PP_TT[9,1,7]=''

Wichtig ist hierbei, dass zwischen den Hochkommas keine Leertaste gemacht werden darf. Eine Leertaste hat programmtechnisch einen eigenen Zeichencode, es würde also abgefragt werden, ob die Variable PP_TT[9,1,7] den Zeichencode der Leertaste enthält. Dies ist aber nicht gleichbedeutend wie die Abfrage, ob die Variable gar nichts enthält!

Mögliche Vergleichsoperatoren:

Operator	Bedeutung
=	gleich
<>	ungleich
>	größer als
<	kleiner als

Beispiel:

Die Prüfung ist «iif(M->PP_TT[9,1,7]='J', 'bestanden', 'nicht bestanden')»

Ergebnis:

Enthält das Feld M->PP_TT[9,1,7] ein J:
Die Prüfung ist bestanden.

Enthält das Feld M->PP_TT[9,1,7] irgendetwas anderes oder auch gar nichts:
Die Prüfung ist nicht bestanden.

Alternativ kann auch diese Formulierung verwendet werden:

Die Prüfung ist «iif(M->PP_TT[9,1,7]<>'J', 'nicht bestanden', 'bestanden')»

Von der Abfragelogik her kommt man hiermit zum gleichen Ergebnis. Zu beachten ist jedoch, dass bei der Ungleich-J-Abfrage der Dann- und der Sonst-Fall umgekehrt sein muss als bei der Gleich-J-Abfrage!

Unter Vorbehalt wäre auch diese Formulierung denkbar:

Die Prüfung ist «iif(M->PP_TT[9,1,7]='N', 'nicht bestanden', 'bestanden')»

Wichtig:
Zu beachten ist hierbei jedoch, dass die jeweils abgefragte Variable dann auch nur J oder N enthalten darf! Besteht die Möglichkeit, dass entweder J, N oder gar nichts enthalten ist, hätte das zur Folge, dass immer dann, wenn nichts enthalten ist, ebenfalls „bestanden“ ausgegeben wird, was ein fataler Fehler wäre! Aus diesem Grund wird diese Formulierung nicht empfohlen. Bei der Gleich-N-Abfrage muss der Dann- und der Sonst-Fall ebenfalls umgekehrt sein als bei der Gleich-J-Abfrage.

Iif-Abfragen mit bedingter Ausgabe von Feldinhalten

«iif(M->PP_TT[9,1,23,1,2,1,20] ='M',M->PP_TT[2,1,4], M->PP_TT[2,1,8])»

In dieser Beispielabfrage wird als Bedingung abgefragt, ob im Feld „Geschlecht“ ein M enthalten ist. Trifft dies zu, wird die Variable PP_TT[2,1,4] für die männliche Bezeichnung des Abschlusses ausgegeben (z. B. Geprüfter Personalkaufmann). Trifft dies nicht zu, wird die Variable PP_TT[2,1,8] für die weibliche Bezeichnung des Abschlusses ausgegeben (z. B. Geprüfte Personalkauffrau).

Wichtig:
Wird im Dann- oder Sonst-Fall eine Umwandlung mit val() vorgenommen, so muss der Datentyp in beiden Fällen gleich sein. Wird also einerseits eine normale Variable oder auch ein Text ausgegeben und im anderen Fall eine mit val() umgewandelte Variable, muss hier der Datentyp korrigiert werden. Die mit val() umgewandelte Variable muss also zusätzlich mit der str()-Formel in Text umgewandelt werden.

Beispiel:

«iif(val(M->PP_TT[9,1,24,1,14])<50, 'nicht bestanden', str(val(M->PP_TT[9,1,24,1,14]),3,0))»

iif-Abfragen mit Ausgabe von verknüpften Feldinhalten

«iif(M->PP_TT[9,1,24,1,6]<>'0', M->PP_TT[9,1,24,1,17] +' Uhr bis '+ M->PP_TT[9,1,24,1,18] +' Uhr', 'bereits bestanden')»

In dieser Funktion wird abgefragt, ob die Variable PP_TT[9,1,24,1,6] (Befreit-Kennzeichen) einen anderen Inhalt als '0' hat. Wenn dies zutrifft, tritt folgende Formulierung in Kraft:

M->PP_TT[9,1,24,1,17] +' Uhr bis '+ M->PP_TT[9,1,24,1,18] +' Uhr'

Hierbei handelt es um zwei Variablen, die die jeweilige Beginn- und Endezeit enthalten. Ergänzt werden diese durch Klartext (z. B. „Uhr bis“). Die verschiedenen Feldinhalte und Texte werden einfach mit einem Pluszeichen aneinander gereiht. Hier ist besonders darauf zu achten, dass die Hochkommas, die den Text kennzeichnen, korrekt gesetzt werden, die Variablen dürfen nicht in Hochkommas gesetzt werden. Wichtig ist auch, dass bei dieser unübersichtlichen Formulierung am Ende das Komma, das den Dann-Fall abschließt, nicht vergessen wird.

In den Text-Passagen müssen alle notwendigen Leertasten, Satzzeichen usw. enthalten sein, dies ist vor allem zu Beginn und Ende der Passagen wichtig, da sonst im Ausdruck später die einzelnen Bestandteile „aneinanderkleben“.

Sollen mehrere Variableninhalte miteinander verknüpft werden, ohne dass Text mit ausgegeben wird, ist die Vorgehensweise genauso:

M->PP_TT[9,1,24,1,17] +' '+ M->PP_TT[9,1,24,1,18]

Hier wird durch +' '+ zwischen den Variablen noch eine Leertaste eingefügt.

Iif-Abfragen mit mehreren Wenn-Bedingungen

«iif(val(M->PP_TT[9,1,24,1,14])<50 and M->PP_TT[12,1,1]='1','Fachbereich (Projektarbeit):','')»

Wie in diesem Beispiel gezeigt wird, ist es auch möglich, mehrere Wenn-Bedingungen miteinander zu verknüpfen. Dies kann sowohl mit „and“ als auch mit „or“ geschehen, je nachdem ob eine und-Verknüpfung oder eine oder-Verknüpfung gebraucht wird.

Im Beispiel oben wird abgefragt, ob die Variable PP_TT[9,1,24,1,14] kleiner als 50 ist. Da es sich hier um einen direkten Zahlenvergleich handelt, muss die Variable mit der Funktion val() vorher in eine Zahl umgewandelt werden.

Verschachtelte Iif-Abfragen

Normale iif-Abfragen decken mit dem Dann- und Sonst-Fall nur zwei Möglichkeiten ab, also z. B. der Prüfling ist entweder weiblich oder männlich. Gibt es aber mehr als zwei Möglichkeiten, z. B. jemand nimmt zum ersten Mal an einer Prüfung teil oder wiederholt zum ersten oder zweiten Mal, genügt dies nicht mehr.

Man kann das Problem natürlich dadurch lösen, dass man mehrere Abfragen ohne Sonst-Fall nacheinander ins Formular setzt:

«iif(substr(M->PP_TT[4],2,1)= '0', 'Prüfung', '')»«iif(substr(M->PP_TT[4],2,1)= '1', '1. Wiederholungsprüfung', '')»«iif(substr(M->PP_TT[4],2,1)= '2', '2. Wiederholungsprüfung', '')»

Dies kann – je nach Fall – jedoch sehr aufwändig werden. Eleganter ist es, solche Abfragen zu verschachteln, d. h. in der ersten iif-Abfrage (hier grau schattiert) tritt an die Stelle des Sonst-Falls eine weitere iif-Abfrage (nicht schattiert):

«iif(substr(M->PP_TT[4],2,1)= '1', '1. Wiederholungsprüfung', iif(substr(M->PP_TT[4],2,1)= '2', '2. Wiederholungsprüfung', 'Prüfung'))»

Wichtig ist, dass man streng darauf achtet, das sämtliche Klammern gesetzt werden. Jede iif()-Funktion, die mit (geöffnet wird, muss auch wieder mit) geschlossen werden. Wird – wie in diesem Beispiel – zusätzlich noch mit anderen Funktionen wie z. B. substr() gearbeitet, kommen hier noch weitere Klammern hinzu, die geöffnet und auch wieder geschlossen werden müssen.

Außerdem werden innerhalb der iif-Abfrage keine weiteren «»-Zeichen für die verschachtelten iif-Abfragen verwendet, wichtig ist jedoch, das »-Zeichen am Ende der ersten iif-Abfrage nicht zu vergessen.

Es können beliebig viele iif-Abfragen verschachtelt werden, es sollte jedoch darauf geachtet werden, dass das Ganze halbwegs überschaubar und nachvollziehbar bleibt.

Temporär existierende Variablen – VarExists()

Siehe auch "Umstellungsprobleme beim Format docx"!

Viele Variablen werden nur dann ausgegeben, wenn das zugehörige Feld in der Datenbank auch gefüllt ist. Beispielsweise wird der Nachname der Person wohl immer als Variable zur Verfügung stehen. Die Variable für den Namenstitel der Person wird jedoch nur dann übergeben, wenn die Person auch tatsächlich einen Titel hat. Fragt man nun im Formular die Variable des Namenstitels ab, so kann dies dazu führen, dass die Abfrage nicht funktioniert, wenn kein Titel vorhanden ist.

Dieses Problem kann dadurch gelöst werden, dass die Ausgabe dieser temporären Variablen in einer Abfrage erfolgt, die zuerst prüft, ob die Variable zum Ausgabezeitpunkt überhaupt vorhanden ist.

Beispiel:
In einem Formular soll – falls vorhanden - die vom Azubi gewählte Wahlqualifikation ausgegeben werden. Zu Lösung wird zunächst eine iif-Abfrage eingefügt. Innerhalb dieser Abfrage wird als Wenn-Bedingung dann die Funktion VarExists() eingefügt:

«iif(VarExists("BB_AZUBI_WAHLQUALI[1,2,1,1]"), M->BB_AZUBI_WAHLQUALI[1,2,1,1], "")»

Die Abfrage lautet also: Wenn die Variable existiert, dann gib diese aus. Wenn sie nicht existiert, wird in diesem Fall nichts getan. Es wäre jedoch auch möglich, einen Dann-Fall einzufügen, der in diesem Falle beispielsweise eine andere Variable ausgibt.

Bedingte Zeilenschaltung mittels iff-Abfrage

Siehe auch "Umstellungsprobleme beim Format docx"!

In manchen Fällen kann es sinnvoll sein, nur dann eine Zeilenschaltung durchführen zu lassen, wenn bestimmte Kriterien erfüllt sind. Dies kann durch eine iif-Abfrage gelöst werden.

In unten stehendem Beispiel wird Beginn- und Endezeit des ersten Prüfungstages ausgegeben, die Endezeit soll dabei in einer neuen Zeile beginnen. Muss der Prüfling nicht zum Termin erscheinen, weil er dieses Prüfungsfach bereits bestanden hat, soll nur „bereits bestanden“ ausgegeben und auf eine weitere Zeilenschaltung verzichtet werden.

Beispiel:

«iif(M->PP_TT[9,1,24,1,6]<>'0', M->PP_TT[9,1,24,1,17] +' Uhr bis
'+ M->PP_TT[9,1,24,1,18] +' Uhr', 'bereits bestanden')»

Zu beachten ist, dass sich die gesamte iif-Abfrage tatsächlich über zwei Zeilen erstreckt, aber trotzdem eine zusammenhängende Funktion bildet.

Soll nur eine Zeilenschaltung ohne weitere Ausgabe von Text gemacht werden, muss die Zeilenschaltung in Hochkommas gesetzt werden!

«iif(M->PP_TT[9,1,24,1,6]<>'0', M->PP_TT[9,1,24,1,17] +'
'+ M->PP_TT[9,1,24,1,18] +' Uhr', 'bereits bestanden')»

Unterdrückung von Leerstellen

Je nach Situation kann es dazu kommen, dass man in einer Situation wahlweise Leertasten benötigt oder auch nicht. Dies kann z. B. der Fall sein, wenn man eine mehrzeilige Anschrift nicht als Anschriftenblock ausgeben möchte, sondern in einer Zeile. Die Anschriften sind so aufgebaut, dass bei kurzen Anschriften die obersten Zeilen leer sind. Würde man nun die Variablen mit Leertasten getrennt aneinanderreihen, bekäme man für jede nicht gefüllte Variable trotzdem die Leertaste ausgedruckt, wodurch im Vergleich mit dem übrigen Text eine Lücke entsteht. Durch die Verwendung der Funktion alltrim() kann dies verhindert werden.

Beispiel:
«alltrim(M->PP_TT[3,2,2,1,13,1,5] + ' ' + M->PP_TT[3,2,2,1,13,1,6] + ' ' + M->PP_TT[3,2,2,1,13,1,7] + ' ' + M->PP_TT[3,2,2,1,13,1,8])»

Die oben gezeigte Lösung kann bei Bedarf natürlich auch in eine iif-Abfrage integriert werden.

Darstellung von Punkten aus Prüfungsergebnissen ohne Kommastellen

In bestimmten Fällen kann es wünschenswert sein, Prüfungsergebnisse ohne Nachkommastellen darzustellen, also statt 70,00 Punkte soll 70 Punkte im Formular stehen.

Um dies zu erreichen, kann eine der beiden Funktionen verwendet werden:

«alltrim(substr(padl(M->PP_TT[9,1,24,1,14],6,' '),1,3))»

«alltrim(str(val(M->PP_TT[9,1,24,1,14]),3,0))»

Wichtig:
Diese Funktionen schneiden die Nachkommastellen lediglich ab, eine Rundung erfolgt nicht!

Problemlösung für Sonderzeichen, z. B. ß

Gelegentlich kann es dazu kommen, dass bestimmte Zeichen in iif-Abfragen zu Problemen führen, wenn Klartext ausgegeben werden soll, weil sie einen Programmfehler verursachen. Das Problem tritt aber nur bei Klartextausgaben auf, ist das Zeichen in einer aus tibros kommenden Variable enthalten, verursacht es keinen Fehler, da hier eine andere Verarbeitung vorliegt.

Konkret tritt das Problem z. B. auf, wenn das Zeichen ß im Klartext ausgegeben wird. In diesem Fall funktioniert die Abfrage nicht, obwohl die korrekte Syntax eingehalten wurde. Diese Problematik lässt sich leider nicht beseitigen, kann aber durch Nutzung eines Steuercodes "umschifft" werden.

Dazu wird die Klartexteingabe durch ein Hochkomma beendet und ein Pluszeichen angefügt. Anschließend wird die Ausgabe eines Steuercodes definiert, indem die Formulierung chr(xxx) eingegeben wird. An Stelle der hier gezeigten xxx muss die jeweilige ASCII-Code-Nummer für das Sonderzeichen angegeben werden. Anschließend wird wieder ein Pluszeichen und ein Hochkomma gemacht und die iif-Abfrage normal fortgesetzt.

Beispiel:

«iif(M-> PP_TT[10,1,7]='01','Schillerstra'+chr(223)+'e 8, 71640 Ludwigsburg','Berliner Platz 8, 71638 Ludwigsburg')»

Im Beispiel soll, wenn der Inhalt der Variable 01 ist, eine bestimmte Anschrift als Klartext ausgegeben werden. Das hier in "Schillerstraße" auftauchende ß würde dazu führen, dass die Abfrage nicht funktioniert. Deshalb wird der Klartext unterbrochen und '+chr(223)+' eingefügt, da 223 der ASCII-Code für das Zeichen ß ist. Danach wird die Abfrage normal fortgesetzt.

Die richtige ASCII-Code-Nummer für das jeweils benötigte Zeichen lässt sich im Internet ermitteln, beispielsweise über http://www.goascii.de/ oder über http://de.wikipedia.org/wiki/ASCII-Tabelle

Umstellungsprobleme beim Format docx

Nicht funktionierende iif-Abfragen

In einigen Fällen kommt es vor, dass iif-Abfragen nicht funktionieren, obwohl die Formel an sich keinen logischen Fehler aufweist. Dies kann unter Umständen dadurch verursacht werden, dass eine der beiden Ausgabebedingungen so formuliert ist, dass in diesem Fall nichts geschehen soll. Normalerweise würde es genügen, in diesem Fall zwei Hochkommas zusetzen:

«iif(M->PP_TT[9,1,7]='J', 'bestanden', '')»

Funktioniert die Formel unter docx nicht, obwohl sonst kein Fehler vorliegt, kann zwischen die Hochkommas ein Leerzeichen eingegeben werden:

«iif(M->PP_TT[9,1,7]='J', 'bestanden', ' ')»

Durch die Ausgabe des Leerzeichens funktioniert die Formel dann wieder

Temporär existierende Variablen – VarExists()

Wird mit VarExists() danach gefragt, ob eine Variable existiert, kann es dazu kommen, dass die iif-Abfrage nicht funktioniert, obwohl kein logischer Fehler vorliegt. Dies ist insbesondere dann der Fall, wenn die Variable nicht existiert und mit diesem Zweig weiter gearbeitet werden soll, z. B. mit einer verschachtelten Abfrage. Eine konkrete Lösung für das Problem konnte noch nicht gefunden werden. Es kann lediglich versucht werden, die Abfragen so umzugestalten dass das Problem umgangen wird.

Bedingte Zeilenschaltung:

Bisher konnte eine bedingte Zeilenschaltung dadurch erreicht werden, dass innerhalb der Formel die Word-Absatzmarke eingefügt wurde. Dies funktioniert nun nicht mehr. Als Ersatz kann die Kombination chr(13)+chr(10) verwendet werden.

Beispiel:
«iif(M-> AZUPRUEF[1,14]='00' or M-> AZUPRUEF[1,14]='01' or M-> AZUPRUEF[1,14]='02',M->azupruef[1,15,1,1]+chr(13)+chr(10),' ')»

Bedingte Tabellen:

Im doc-Format war es möglich, innerhalb einer iif-Abfrage einen Tabulatorpfeil zu verwenden, um das ausgegebene Ergebnis in Tabellenform zu gliedern oder größere Abstände einzufügen. Dies ist im Format docx nicht mehr möglich. Hier müssen die Formeln so umgestaltet werden, dass eine anderweitige Formatierung, beispielsweise über eine fest eingefügte Tabelle, möglich ist.

Beispiel:

Alt:

```
¶
«iif(M->azubi[1,9]='M','Herr','Frau')»·«alltrim(trim(M->azubi[1,26])+'·'+M->azubi[1,20])»·«M->azubi[1,19]»·ist·für·folgende·Prüfungstermine·vorgesehen:¶
¶
«iif(M->·AZUPRUEF[1,14]='00'·or·M->·AZUPRUEF[1,14]='01'·or·M->·
AZUPRUEF[1,14]='02',M->azupruef[1,9,1,1]+':→'+M->azupruef[1,15,1,1],'')»¶
«iif(M->·AZUPRUEF[2,14]='00'·or·M->·AZUPRUEF[2,14]='01'·or·M->·
AZUPRUEF[2,14]='02',M->azupruef[2,9,1,1]+':→'+M->azupruef[2,15,1,1],'')»¶
«iif(M->·AZUPRUEF[3,14]='00'·or·M->·AZUPRUEF[3,14]='01'·or·M->·
AZUPRUEF[3,14]='02',M->azupruef[3,9,1,1]+':→'+M->azupruef[3,15,1,1],'')»¶
«iif(M->·AZUPRUEF[4,14]='00'·or·M->·AZUPRUEF[4,14]='01'·or·M->·
AZUPRUEF[4,14]='02',M->azupruef[4,9,1,1]+':→'+M->azupruef[4,15,1,1],'')»¶
¶
```

Neu:

Die·voraussichtlichen·Prüfungstermine·sind:¶

«iif(M->·AZUPRUEF[1,14]='00'·or·M->·AZUPRUEF[1,14]='01'·or·M->·AZUPRUEF[1,14]='02',M->azu-pruef[1,9,1,1]+':'+chr(13)+chr(10),'·')»«iif(M->·AZUPRUEF[2,14]='00'·or·M->·AZUPRUEF[2,14]='01'·or·M->·AZUPRUEF[2,14]='02',M->azu-pruef[2,9,1,1]+':'+chr(13)+chr(10),'')»«iif(M->·AZUPRUEF[3,14]='00'·or·M->·AZUPRUEF[3,14]='01'·or·M->·AZUPRUEF[3,14]='02',M->azu-pruef[3,9,1,1]+':'+chr(13)+chr(10),'·')»«iif(M->·AZUPRUEF[4,14]='00'·or·M->·AZUPRUEF[4,14]='01'·or·M->·AZUPRUEF[4,14]='02',M->azu-pruef[4,9,1,1]+':'+chr(13)+chr(10),'·')»¤	«iif(M->·AZUPRUEF[1,14]='00'·or·M->·AZUPRUEF[1,14]='01'·or·M->·AZUPRUEF[1,14]='02',M->azupruef[1,15,1,1]+chr(13)+chr(10),'·')»«iif(M->·AZUPRUEF[2,14]='00'·or·M->·AZUPRUEF[2,14]='01'·or·M->·AZUPRUEF[2,14]='02',M->azupruef[2,15,1,1]+chr(13)+chr(10),'·')»«iif(M->·AZUPRUEF[3,14]='00'·or·M->·AZUPRUEF[3,14]='01'·or·M->·AZUPRUEF[3,14]='02',M->azupruef[3,15,1,1]+·chr(13)+chr(10),'')»«iif(M->·AZUPRUEF[4,14]='00'·or·M->·AZUPRUEF[4,14]='01'·or·M->·AZUPRUEF[4,14]='02',M->azupruef[4,15,1,1]+·chr(13)+chr(10),'')»¤

¶