

ModPar Version 2.57

Modulare Parameter für Allegro-C/a99

Inhaltsverzeichnis

1. Allgemeines.....	5
2. Bedingungen und Vereinbarungen.....	6
2.1. allgemeine Festlegungen.....	6
2.2. Bezeichnungskonventionen.....	6
2.3. Kategorie-Schema.....	7
2.3.1. Allgemeines.....	7
2.3.2. Änderungen und Abweichungen in den Kategorien zur Katalogisierung.....	7
3. Zentrale Funktionen.....	9
3.1. Bibliotheken.....	9
3.2. Initialisierung beim Programmstart.....	10
3.3. Einbindung von zusätzlichen Initialisierungen beim Programmstart.....	11
3.4. Die dynamischen Menüs.....	11
3.5. Standard-Mechanismen.....	17
3.5.1. Satz bearbeiten.....	17
3.5.2. Satz löschen.....	17
3.5.3. Neusatz.....	17
3.5.4. Satz kopieren.....	17
3.5.5. Offline-Speicherbehandlung.....	18
3.5.6. temporäre Dateien.....	18
3.5.7. private Dateien / Arbeitsverzeichnis.....	18

3.5.8. datenbankspezifische Dokumentationen u. Hilfedateien	18
3.5.9. Auswahl der Flexdateien für die Belegung der Buttons.	19
4. Satzarten.....	20
4.1. Satzarten für bibliographische Daten.....	20
4.2. Satzarten für Lokaldaten.....	21
4.3. Satzarten für Stammsätze mit bibliothekarischem Inhalt.....	21
4.4. Satzarten für Organisationsdaten.....	22
4.5. Satzarten für Konfigurationsdaten.....	23
4.5.1. Konfigurationssätze.....	23
4.5.2. Generatorsätze.....	24
5. Globale Variablen zu Steuerung der ModPar-Komponenten.....	25
6. Konventionen zur Programmierung.....	28
6.1. Variablen.....	28
6.2. Unterprogramme.....	28
6.3. Verzeichnisse.....	28
7. Verfahren zur Programmaktualisierung.....	30
7.1. Aktualisierungswege.....	30
7.2. allgemeine Konventionen zum Aktualisierungsverfahren.....	30
8. Funktionsbibliotheken.....	32
8.1. MP_Misc.FLB (allgemeine Hilfsfunktionen).....	32
:MP_Log.....	32
:a99_Restart.....	32
:a99_CheckRestart.....	32
:MP_FileFind.....	33
:MP_FilesDel.....	33
:MP_Var0Del.....	33
:MP_Var1Del.....	33
:MP_CompareEx.....	34
:MP_VL_Init.....	34
:MP_VL_Add.....	34
:MP_VL_Write.....	34
:MP_VL_Show.....	36
:MP_VL_Reset.....	36
:MP_Increment.....	36
:MP_Mkdir.....	36
:MP_EscBackslash.....	37
:MP_SetSignal.....	37
:MP_ResetSignal.....	37

:MP_FlipSet.....	37
:MP_Ver.....	38
:MP_GetTempFileName.....	38
:MP_ListAdjust.....	38
:MP_Trace.....	38
:MP_IdCntInc().....	38
:MP_FetchEx.....	39
:MP_Sound.....	39
:MP_Message.....	40
8.2. MP_ACC.FLB (Berechtigungs-Funktionen).....	40
:MP_AccCheck().....	40
8.3. MP_DaTim.FLB (Datums- & Zeitberechnungen).....	41
:MP_Date2Sort.....	41
:MP_Sort2Date.....	41
:MP_IsDate.....	41
:MP_TimeDiff.....	42
:MP_TimeDiffSec.....	42
:MP_Time2Sec.....	42
:MP_Sec2Time.....	42
:MP_PercRemain.....	43
:MP_TimeRemain.....	43
8.4. MP_Stack.flb (Stackfunktionen).....	44
:PushRec.....	44
:PopRec.....	44
:DropRec.....	44
:PushAdr.....	44
:Return.....	45
:Push.....	47
:Pop.....	47
8.5. MP_DB.FLB (Datenbank-Funktionen).....	48
:MP_CfgGetValue.....	48
:MP_ChkReg.....	48
:MP_Choose().....	48
:MP_Rec2EM.....	49
:MP_CloseActRes.....	49
:MP_RegisterCount.....	49
:MP_FndRecLock.....	50
:MP_Lock.....	50

:MP_PutUnlock.....	50
:MP_FormLoad.....	50
:MP_FormRelease.....	51
:MP_FormProc.....	51
:MP_QTExpand().....	51
:MP_ResultSort.....	52
:MP_FindFirst.....	52
8.6. MP_Dis.FLB (Anzeigefunktionen).....	54
:MP_Dis.....	54
:MP_DSet.....	54
:MP_DisClear.....	54
:MP_DisAdd.....	54
:MP_DisAddLine.....	54
:MP_DisAddLineHidden.....	55
:MP_DisShow.....	55
:MP_SfMes.....	55
:MP_UifTxtGet.....	56
8.7. MP_FuncList.FLB (dynamische Menü's).....	57
:MP_FuncList().....	57
:MP_MenuShowLast.....	62
:MP_MenuStackClear.....	62
8.8. MP_Res.FLB (Ergebnismengen-Bearbeitung).....	63
:MP_ResSortEx().....	63
9. Hilfsroutinen und -programme.....	64
9.1. Cover-Verlinkung.....	64
9.2. URL-Darstellung und Verlinkung.....	64
9.3. Verwendung externer Hilfsprogramme.....	64
10. Integration anwenderspezifischer Anpassungen.....	66
10.1. Zweck der Integration anwenderspezifischer Anpassungen.....	66
10.2. Anpassung der Exemplarerfassung.....	66
10.3. Anpassung der Statistik.....	66
Anhang A: Installation der "Modularen Parameter".....	67

1. Allgemeines

"ModPar" steht für modulare Parameter für a99. Die Entwicklung erfolgte, um eine leicht zu pflegenden und zu benutzende Basis von Funktionen für die Nutzung in komplexeren a99-Flex-Anwendungen zu schaffen, die einen leichten Austausch von Modulen und nebenwirkungsarme Anwender-Anpassungen ermöglichen.

Die Modularisierung erfolgt dabei

- über Trennung in Funktionsgruppen,
- durch dynamisch erzeugte, kontextabhängige Menüs mit Zusammenfassung der Funktionsarten in den Menüs über Regeln der Dateinamen-Struktur,
- durch Trennung der Anzeigeparameter nach Satzarten,
- über satzartspezifische Funktionsflexe für Neuanlage, Bearbeiten und Löschen der Satzarten,
- Funktionen zur Internationalisierung,
- und Routinen für automatische Updates

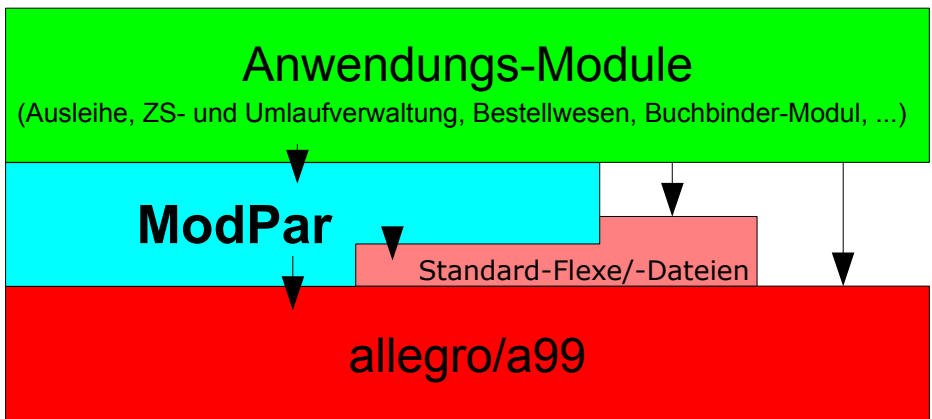


Bild 1: SW-Schichten für Anwendungen mit Allegro / Einordnung der modularen Parameter

2. Bedingungen und Vereinbarungen

2.1. allgemeine Festlegungen

- Cockpit und Presto (DOS) werden nicht mehr verwendet.
- Der Aufruf der Neuerfassung über das Menü „**Bearbeiten**“ - „**Neuaufnahme**“ wird nicht verwendet, da über diesen fest codierte Befehle ausgeführt werden, die nicht zum Konzept der Modularen Parameter passen. Es werden nur die alternativen Aufrufe über **<F9>** oder **[Neusatz(F9)]** genutzt
- Der PV-Abschnitt (programmierbare Validierungen) der Indexparameter wird nicht genutzt; die dort sonst realisierten Funktionen werden über satzspezifische Flexe umgesetzt.
- Im Netzwerkbetrieb liegt das Programmverzeichnis ({PR}) auf dem gemeinsamen share bzw. volume; lokal werden nur die Schriften installiert.
- Zu ModPar gehörende Dateien werden im oder unterhalb des Datenbankverzeichnisses abgelegt, wodurch diese Vorrang gegenüber gleichnamigen Standard-Dateien haben.
- \$-Variablen mit einem Kleinbuchstaben als Namen (z.B. \$t) werden in Flex-Dateien als flüchtige, temporäre Variablen „für zwischendurch“ verwendet, „überleben“ keine UP-Aufrufe oder includes und werden in Dokumentationen nicht gesondert erwähnt.

2.2. Bezeichnungskonventionen

{DB}	Datenbankverzeichnis; z.B. c:\allegro\demo2\
{PR}	Programmverzeichnis; z.B. c:\allegro\
{FX}	Flex-Verzeichnis; z.B. c:\allegro\flex\
{HP}	Hilfdatei-Verzeichnis; z.B. c:\allegro\help\
{HT}	Html-Hilfdateien; z.B. c:\allegro\html\
{BIN}	Tools-Verzeichnis z.B. c:\allegro\mp_bin\

Funktionsnamen, die auf '()' enden, werden über den Stack aufgerufen
(z.B. 'MP_Choose()', s.a. Bibliothek MP_Stack)

2.3. Kategorie-Schema

2.3.1. Allgemeines

Die modularen Parameter unterstützen das \$A-Schema der UB Braunschweig (B. Eversberg) in der Fassung vom 11.11.2004, das sogenannte "konsolidierte Schema".

Für Funktionsmodule wie Ausleihe, und Erwerbung werden die Definitionen von Herr Hartwig, die ursprünglich für DOS-ALF und DOS-Order erstellt wurden, unterstützt.

Für die neueren Entwicklungen, insbesondere für die Periodika- und Umlaufverwaltung, finden erweiterte und/oder geänderte Satzarten, Kategorien und Geschäftsregeln Anwendung.

2.3.2. Änderungen und Abweichungen in den Kategorien zur Katalogisierung

Abweichungen gegenüber \$A-Schema 2004:

Inhalt	RDA	MAR C	V36ff	ModPar	Bemerkungen
URL des Cover-Bildes			???	#94c	die #94 wird in der ausgelieferten V36 nicht genutzt

Erfassung RDA-relevanter Inhalte

Inhalt	RDA-Nr.	MARC	Kategorie V36ff	Kategorie ModPar
erfasst nach		040__\$e	#10	#10
Inhaltstyp		336	#31f	#31f
Medientyp		337\$a \$b \$2	-	#770 \$a \$b \$2
Datenträgertyp		338 \$a \$b	#77.1 (wird z.Zt.	#771

Inhalt	RDA-Nr.	MARC	Kategorie V36ff	Kategorie ModPar
			vom import in #77c geschrieben)	
Beziehungskennzeichnung		xxx__\$e	noch keine Festlegung (08/2016)	\$e der jeweiligen Personen- oder Körperschaftskategorie
Verantwortlichkeitsangabe		245__\$c	#39	#39
Text-Sprache des Originals		041__\$h	#37o	#37o
Herstellung (unveröffentlicht)		264_0	(?)	#730 \$a\$b\$c
Veröffentlichung		264_1	#74...#76	#74 ...#76
Verteilung		264_2	#73.2	#732 \$a\$b\$c
Druck u./oder Vervielfältigung		264_3	#73.3	#733 \$a\$b\$c
Copyright		264_4	#73.4	#734 u. #76c aus \$c

Hinweis zur Beziehungskennzeichnung:

Die Verfasserkategorien #40. dienen für nach RDA erfasste Sätzen als Container für alle geistigen Schöpfer; alle anderen Personen Körperschaften werden in den Kategorien ab #42 bzw. #61 erfasst. Die Teilfelder werden 1:1 aus dem MARC-Standard übernommen.

Beispiel: #40 Tester, Otto\$d1959-\$eVerfasser\$4aut

3. Zentrale Funktionen

Zentrale Funktionen werden über folgende Dateien bereitgestellt:

3.1. Bibliotheken

MP_DB.flb	Datenbank- und Formular-Funktionen
MP_Dis.flb	Anzeigefunktionen
MP_Misc.flb	diverse Hilfsfunktionen
MP_DaTim.flb	Datums- und Zeitfunktionen
MP_Stack.flb	stellt Funktionen für schachtelbare UP's sowie diverse Stackfunktionen für allgemeine Zwecke zur Verfügung
MP_Janas.flb	Anzeige von template-gesteuerten Explorerfenstern über MP_ShowJanas()
MP_Satztyp.flb	MP_RecordTypGet() ermittelt die aktuelle Satzart MP_IsRelation() ermittelt die Satzhierarchie
MP_Mail.flb	Funktionen zum Versenden von Mails
MP_UIF.flb	MP_UifTxtGet() ermittelt Text über Texcode und Sprachdatei
MP_FktLst.flb	enthält Komponenten zur Menüsteuerung
MP_Struct.flb	enthält Funktionen zur Verwaltung von Kennungen, z.B. von Medientypen
MP_Res.flb	Funktionen zur Ergebnismengenbehandlung

weitere Dateien:

_start.flx	ermittelt DB-Parameter aus Systemsatz, bildet Prozess-ID, wenn von akt. a99-Version nicht unterstützt und loggt den Programmstart
_endflx.flx	loggt u.a. das Programmende
mp-chain.flx	Chaining-Funktion für Flex-Dateien
onprop.flx	zeigt Eigenschaften des aktuellen Satzes und der aktuellen Datenbank
mp-care-rec.flx	sperrt/entsperrt des aktuell angezeigten Satzes

Realisiert werden damit u.a.:

- Menüsteuerung
- Anzeigefunktionen
- Datums- und Zeitberechnungen
- Logging, Datensatznummern-Stack, Registerzugriff, Stringvergleiche
- Lesen, Verriegeln/Entriegeln und Speichern von Datensätzen
- a99-Neustarts mit Wiederherstellung der letzten Ansicht
- gruppenweises Löschen von \$-Variablen

3.2. Initialisierung beim Programmstart

Bei der Initialisierung werden folgende Schritte abgearbeitet:

- Prüfung des Programm- und Datenbank-Pfades auf enthaltene Tiefstriche und Punkte
- Erzeugung einer Prozess-ID und deren Speicherung in #uPI
- Setzen der #uOp aus #op oder Ersatzweise aus der Umgebungsvariablen 'USERNAME'
- Setzen des Arbeits- und Temp-Verzeichnisses auf Unterverzeichnisse des DB-Verzeichnisses
- Anpassung der a99-Menüs je nach vorliegender Windows-Version
- Vermerk des a99-Starts in der log_1st.log
- Test auf unterstützte a99-Versionen, gegebenenfalls Warnung oder Abbruch
- Prüfung Werte SaveResults und SaveAsk in der verwendeten ini-Datei; Sollwert ist jeweils "2", ansonsten Ausgabe einer Warnung
- Einlesen der Datenbank-Konfiguration
- Einlesen der privaten DB-Konfiguration
- wenn Zweigstellen-Kürzel hinterlegt ist, passende Zweigstellenkonfiguration einlesen (ab V2.41)
- einen Datensatz aus der Datenbank lesen, um interne Initialisierungen zu provozieren
- dummy-qrix-Befehl ausführen aus gleichem Grund
- im Arbeits- und Temp-Verzeichnis evtl. verbliebene *.vw und *.vwn-Dateien löschen
- Viewliste für Medientypstammsätze neu erzeugen
- Anwender-Schaltflächen mit Standard-Werten belegen
- wenn vorhanden, private Schaltflächenbelegung herstellen

- Funktions-Menü in Menüleiste einblenden
- Prüfen, ob a99-Start Folge eines Rstarts war; wenn ja, zwischengespeicherte Variablen einlesen und hinterlegten Start-Befehl ausführen
- in Konfiguration hinterlegten Start-Helptext anzeigen
- optional vorhandene Zusatz-Initialisierungen ausführen

3.3. Einbindung von zusätzlichen Initialisierungen beim Programmstart

Im Systemsatz (Datenbank-Konfiguration, lokale Konfiguration) wird im jeweiligen Teilfeld \$I eine Liste der init-Flexe abgelegt.

Diese werden am Ende der `_start.flx` an `mp-chain.flx` übergeben und damit abgearbeitet.

Jedes dieser Init-Flexe muss dazu mit der Anweisung "exec X mp-chain" enden.

3.4. Die dynamischen Menüs

Die Auswahl der Inhalte für die dynamischen Menüs erfolgt nach:

- Bestandteilen des Dateinames der FLX-Dateien,
- Art des Aufrufs (Parameter)
- Merkmalen des gerade angezeigten Datensatzes
- aktuellen Rechten des angemeldeten Benutzers

Sortiert werden die Menüzeilen nach

- dem in den FLX-Dateien abgelegten SRT-Parameter,
- nach dem Dateinamen (bei gleichem Sortierparameter)

Die Steuerung der Menüs wird über folgende Dateien realisiert:

`mp-fkt.flx` startet über `MP_FuncList()` aus `MP_FktLst.flb` das Menü

onerror.flx	startet mp-fkt.flx mit dem Inhalt des Schreibfelds
onkontext.flx	startet mit dem über MP_RecordTypGet ermittelten Satztyp-Kürzel mp-fkt.flx zur Anzeige des Menüs
onf8.flx	startet onkontext.flx (nicht unter alcarta)
onhelp.flx	realisiert den Aufruf von menüpunktspezifischen Hilfe-dateien

Es werden folgende Szenarien für die Funktionsauswahl unterstützt;

- A) alle Funktionen, deren Aktivierungsbedingung erfüllt ist, werden angeboten
- B) alle Funktionen wie A) werden angeboten, die außerdem im Dateinamen den Teilstring (kürzel) enthalten

Der Inhalt jeder Flexdatei in {DB} wird zeilenweise ausgewertet, bis eine Leerzeile oder die Zeichenkombination @@ auftritt.

Es werden alle Zeilen interpretiert, die einen String XXX: enthalten:

XXX Bedeutung/Syntax

CHR: Char-Set, in dem die Beschreibungen und Texte codiert sind.
Werte: ansi oder ascii
default: ascii

FKT: Zeile enthält die Bezeichnung der Funktion so, wie diese in der angebotenen Auswahl angezeigt werden soll

HELP: (rtf-Datei)|(html-Datei)
wird im Menü über dem jeweiligen Funktionseintrag angezeigt

für Anzeige: rtf-Datei (ohne Endung angeben)
für janas: html-Datei (mit Endung)
Beispiel: HELP: Name.html
oder HELP: Name

PAR: Parameter, der der Funktions-Flexdatei übergeben werden soll

SRT: Wert, nach dem die Einträge der fertigen Liste sortiert werden
 Wertebereich: (0 ... 65535)
 Fehlt SRT, gilt SRT: 0.

Folgende Bereiche dienen der Gliederung nach Funktionsgruppen (ab MP V3.x):

von	bis	Bereich/Gruppe
0	499	Hilfe 400 usr-Hilfen
500	999	Service- (unterstützende Hilfsfunktionen) 600 Mail
1000	1999	Katalogisierung
2000	2999	Export 2001 Neuerwerbung 2500 Listen 2600 Schriftverkehr
3000	3999	Import
4000	4999	Statistiken/Auswertungen
5000	5999	Ausleihfunktionen 5200 Mahnungen 5300 Kasse
6000	6999	Bestellfunktionen
7000	7499	ZS-Verwaltung/Abos
7500	7999	Umlaufverwaltung
8000	8999	Einstellungen 8000 z.B. private Parameter 8800 WebOPAC 8900 Zweigstellen
9000	9999	Systemfunktionen 9000 System-Konfiguration

9900 Zweigstellen
10000 10999 Anwender-Prüffunktionen
11000 11999 System-Prüffunktionen
11950 Index-Checks

MNU: Kontextmenü bedingt unterdrücken

MNU: 0 Kontextmenü verbieten, wenn nur eine Auswahl
mgl.

MNU: 1 Kontextmenü erzwingen, wenn nur eine Auswahl
mgl.

default: durch globale Variable gesteuert

ACC: mindestens erforderlicher Berechtigungs-Level

REQ: Bedingungen, die für die Aufnahme der Funktion in die Auswahl
erfüllt sein müssen. Eine Bedingung kann wie folgt definiert
werden:

`[!]{(feld)}[$(teilstreng)][(op)(teilstreng)](if-bedingung)}`

(feld) Kategorie oder Variable, z.B: #20, #u11, \$xyz

(teilstreng) Teilfeldzeichen

(op) Operator: = - beginnt mit
% - enthält

(teilstreng) Zeichenkette, die in der Kategorie bzw. in deren
Teilfeld gesucht werden soll; darf keine Leer-
zeichen enthalten

(if-bedingung) unterstützt wird: main, New,
hasAuf[@({registernummer}){register}],
hasVol[@({registernummer}){register}],
fe{dir}={Dateiname}
{dir} kann sein: P, D, W oder M, oder
B – Tools-Verzeichnis ({BIN})
F – Flex-Verzeichnis ({FX})
H – Help-Verzeichnis ({HP})

L - Html-Verzeichnis

ChkReg@{cstring} der mit var in einen Parameter für MP_ChkReg umgewandelt werden kann

Beispiel:

ChkReg@#00(e"+") 64 124 "9" 64 "1"
ergibt {teil-id aus #00}@|9@1

in cstring dürfen nicht vorkommen:

[;=@ \x09\x0d\x0a] (durch codes ersetzen)

EM Ergebnismenge muss vorhanden sein

Mehrere Bedingungen werden durch ';' (AND) oder '|' (OR) getrennt.

Die Funktion wird nur aufgelistet, wenn die definierte(n) Bedingung(en) durch den aktuell geladenen Satz erfüllt ist/sind.

Rang der Operatoren (aufsteigend): |;!=%

Leerzeichen sind im Ausdruck nicht erlaubt, Klammerungen werden

nicht unterstützt.

Beispiele: #9B muß vorhanden sein und mit 0 oder 1 beginnen:

REQ: #9B=0|#9B=1

#9B muß vorhanden sein, darf aber nicht mit 2 beginnen:

REQ: #9B;!#9B=2

Im Satz solle #00 und #20 oder #00 und #19 enthalten sein

REQ: #20;#00|#19;#00

Test auf verknüpft gespeicherten Aufsatz

(* in #00):

REQ: #00%*

Einschränkung auf Suchkürzel: Da ohne (kürzel) alle Flexe

aktiviert werden, deren Bedingung erfüllt ist, werden u.U. zu viele Funktionen angeboten.

Damit ein Funktionsflex nur angeboten wird, wenn ein (kürzel) als Parameter angegeben wurde, kann mittels REQ:

`#uÆp=(kürzel)`

genauer eingegrenzt werden.

Beispiel: `REQ: #uÆp=new | wahr, wenn als Such(kürzel) | 'new' angegeben wurde`

Mehrere REQ:-Zeilen werden aneinandergehängt. Folge-Zeilen müssen demzufolge mit "|" oder ";" beginnen.

Kommentare:

Zeilen, die mit "/" beginnen, wirken als Leerzeile und beenden die Auswertung der aktuellen Flexdatei

Besonderheit vor a99, V27.2:

In Anzeigeparametern muss ein Abschnitt #-i (236 dez) enthalten sein, der die Umwandlung der Var. #uÆp in Kleinschreibung realisiert:

```
;----- Abschnitt zur Umcodierung, verwendet in MP_FuncList -----  
#-i  
!uÆp y2 dÆp aÆp           // Æ = Code 146dez  
#+#
```

3.5. Standard-Mechanismen

3.5.1. Satz bearbeiten

In ONFORMS.FLX wird ein Flex {satztyp}-edt.flx gesucht und ausgeführt. Der Satztyp {satztyp} wird aus #0s des Datensatzes entnommen oder über die Funktion MP_RecordTypGet (MP_Satztyp.flb) anhand der Kategorien des aktuellen Satzes ermittelt.

3.5.2. Satz löschen

In ONERASE.FLX wird ein Flex {satztyp}-del.flx gesucht und ausgeführt. Der Satztyp {satztyp} wird aus #0s des Datensatzes entnommen oder über die Funktion MP_RecordTypGet (MP_Satztyp.flb) ermittelt.

3.5.3. Neusatz

In ONNEW.FLX wird ein Menü erzeugt, das seine Einträge aus allen passenden *-new.flx- Dateien erhält. Nicht zum aktuellen Kontext passende Einträge werden dabei unterdrückt (nicht angezeigt).

So wird z.B. keine Exemplar-Neuanlage angeboten, wenn gerade ein Systemsatz in der Anzeige steht.

Ab V2.45: Über die private Datenbank-Konfiguration kann die Dateinummer für Neusätze für jeden Programm-Anwender spezifisch festgelegt werden (Anwender-Code F:nnn;). Diese Einstellung überschreibt dann die Standard-Nummer, die aus der Satzartkonfiguration übernommen wird)

3.5.4. Satz kopieren

In copy-new.flx (über onnew.flx gestartet) wird eine zum Satztyp passende Datei *-copy.flx gesucht und ausgeführt. Gibt es keine solche Datei, wird keine Satzkopie ausgeführt.

3.5.5. Offline-Speicherbehandlung

Nach jeder Lösch- und Speicherfunktion wird der Offline-Speicher geleert, damit es nicht zu "Effekten" kommt, die insbesondere bei Nutzung der 'set obj'-Befehle entstehen konnten (Sätze wurden ohne erkennbaren Grund gelöscht oder verändert, Stand: a99 V31.01)

Diese Funktion ist über den globalen Datenbank-Konfigurationssatz abschaltbar.

3.5.6. temporäre Dateien

Beim Programmstart wird über `_start.flx` ein Verzeichnis `{DB}temp\{operator}` unterhalb des Datenbankverzeichnisses angelegt, das für alle temporären, nicht aufzuhebenden Dateien genutzt wird. Der Inhalt des temp-Verzeichnisses wird zu Beginn (in `_start.flx`) gelöscht. [1]

3.5.7. private Dateien / Arbeitsverzeichnis

Unterhalb `{DB}` wird analog zum Temp-Verzeichnis ein Verzeichnis `dbaux\{operator}` angelegt, das als Arbeitsverzeichnis benutzt wird. Dateien in diesem Verzeichnis bleiben jedoch nach Verlassen und Neustart des Programms erhalten. [1]

3.5.8. datenbankspezifische Dokumentationen u. Hilfedateien

`{DB}updates` hier werden DB-spezifische Beschreibungen abgelegt

`{DB}backups` vor einem Neuaufbau der Indexdateien werden hier die alten `?8?`-Dateien (Grunddatei-Sicherheitskopien) als zip-Archive archiviert.

3.5.9. Auswahl der Flexdateien für die Belegung der Buttons

Mit einem Klick mit der rechten Maustaste auf eine Anwender-Schaltfläche („Button“) kann man eine Funktion, die durch diese ausgelöst werden soll, auswählen.

Damit Flex-Funktionen in der Liste der einer Anwender-Schaltfläche ("Button") zuordenbaren Funktionen erscheinen, müssen Sie vor der ersten Leerzeile einen Eintrag folgender Form enthalten:

BTN: Aufschrift|Tooltip;Parameter

Die 'Aufschrift' erscheint auf der Schaltfläche, 'Parameter' wird der Flexfunktion in der iV bei Aufruf als solcher übermittelt.

'Aufschrift', 'Tooltip' und 'Parameter'-Texte müssen ansi-codiert sein.

4. Satzarten

In jedem Datensatz wird in der Kategorie #0s ein Kürzel abgelegt, das die Satzart beschreibt.

Diese Kürzel werden zur Steuerung der Bearbeitungsfunktionen (onforms.flx) und der Anzeige genutzt.

4.1. Satzarten für bibliographische Daten

TIT	Titelsatz (Buchmedien)
	g Gesamtaufnahme (Überordnung bei mehrb. Werken)
	h Hauptaufnahme (ohne Unterordnungen)
	y mittl. Hierarchie (es folgen u- oder weitere y-Sätze)
	u Untersatz (unterste Hierarchie)
MOV	Film auf Film, Videokassette, CD-I, DVD
ZSS	ZS-Stammsatz
ZSV	ZS-Volume Jahrgang; im Jahrgang können gebundene und ungebundene Hefte nebeneinander bestehen
ZSG	ZS-Band gebundene Hefte
ZSH	ZS-Heft einzelnes, ungebundenes Heft
GKD	Körperschafts- stamm
PND	Personenstamm
PUB	Verlagsstamm

SWD		Schlagwortstamm	
AUF		Aufsatzenth. Werk, unselbständig; d.h. hierzu KANN es keine Exemplare geben! auch: Artikel, Beitrag, usw.	
STF		Festival-Material mehrere, nicht nachweisen	Sammelaufnahme; kann getrennt erfasste Medien
LOS		Loseblatt-Werk	#81p mit L besetzt
LBE	Loseblatt-	#92d besetzt Ergänzungslieferung	

4.2. Satzarten für Lokaldaten

EXS		Exemplarsatz Untergliederung)	(keine weitere
ORD		Bestellsatz	
VEN		Lieferantenstamm	

4.3. Satzarten für Stammsätze mit bibliothekarischem Inhalt

LEX	Lexikon	#96b besetzt	
THE		Thesaurus	
CLA		Klassifikationsstamm	
FST		Festival-Stamm	Stammsatz für Festival- Informationen

REF Verweisungen #9s - allgem. Verweisungen

4.4. Satzarten für Organisationsdaten

TXT	Textbaustein	früher: #9A XCode
	sip	für AV-SIP2
	ord	für Bestellsystem
	m	Mahntexte (s.a. #9A XML...)
HLP	Hilfetext	
DIS	Verteiler	
	N	Normal (wie Stern, aber mit Verbleib)
	s	Stern (n Ex. an je einen Entleiher)
	u	Umlauf (1 Ex. an n Entleiher
		nacheinander; evtl. mit Verbleib beim letzten
		Entleiher)
	i	Info: Information an Entleiher über
		Eintreffen des Mediums
KAL	Kalender	
STA	Statistiken	
KAS	Kassensatz	
MEM	Memo-Satz	#9a besetzt
CUR	Währungstabelle	
QUO	Kontingent	
PUR	Auftraggeber	
SLK	Leserklasse	
GEB	Gebühren	
MTP	Medientyp	
MTS	Medientypstamm	

4.5. Satzarten für Konfigurationsdaten

4.5.1. Konfigurationssätze

Satztyp: CFG

#9A c{code} **Konfigurationssätze**

code ist modulspezifisch, z.B.

DB Datenbank allgemein (ModPar-Grundeinstellungen)

FL modulspezifische Konfiguration FlexLend

FO modulspezifische Konfiguration FlexOr

FP modulspezifische Konfiguration FlexPer

P Druckkonfiguration allgemein

R Parameter Quittungsdruck

RT **Record-Typ**

#9A cRT(recordtyp-Kürzel)

\$N Klartextbezeichnung

\$A Access-Level

\$F Vorzugs-Dateinummer

\$K mit ; getrennte Liste zugehöriger
Kategorienummern

SIP2 SIP2-Schnittstelle

U Umlauf (FlexPer)

OPAC Konfiguration AndoLib-OPAC (ab OPAC V3.1,
ModPar V2.45)

#9A\$\$ cOPAC

#9A\$N Katalog-Überschrift für OPAC

#9A\$C Inhaltliche Beschreibung (code 20 =
Zeilenwechsel)

4.5.2. Generatorsätze

Satztyp: GEN

#9A G{kenn} #9A G(kenn)\$N(name)\$Z(wert)\$z(doku)\$C(code)

ersetzt Z-, S- und B-Generatorsätze der Standard-Konfiguration (\$A-Schema, z.B. #9A GZSTD statt #9A ZSTD).

Über den im Satz Teilfeld C gespeicherten Flex-Code ist ein beliebiges Verfahren zum Bilden von Kennungen (nicht nur Nummern) möglich.

In eigenen Flex-Scripten ist eine transparente Nutzung über die Funktion MP_GenNxtNum() möglich.

Zur Bedeutung der Teilfelder:

kenn	zur Unterscheidung verschiedener Generatorsätze (Kennung);	vergeben	sind:
	- E - Barcodes		
	- Z - Zugangsnummern		
	- S - Signaturen		
	- L - Leser		
	- V - Lieferanten		
	- R - Rechnungen		
name	Klartext-Beschreibung		
wert	letzter vergebener Wert		
code	Code zur Ermittlung des nächsten Wertes		
	in allegro-Exportsprache;		
	Codierungen:		
	- # als %23		
	- " als %22		
	- Teilfeld-Zeichen als %1f		
doku	Beschreibung der Funktion (Hilfetext)		

5. Globale Variablen zu Steuerung der ModPar-Komponenten

#u/EY

Speicher für Inhalt des DB-Konfigurationssatzes;
gesetzt in _start.flx aus System-Konfigurationssatz
benutzte Teilfelder:

TF	Bedeutung	privat

\$V	ModPar-Version aus version-mp	
\$K	Katalogname	
\$N	Bibliotheksadr. 1. Zeile	x
\$n	Bibliotheksadr. 2. Zeile	x
\$s	Bibliotheksadr. Straße	x
\$p	Bibliotheksadr. PLZ Ort	x
\$m	Bibliotheksadr. E-Mail	x
\$t	Bibliotheksadr. Telefon	x
\$x	Bibliotheksadr. Fax	x
\$c	Absenderzeile Adressfeld	x
\$B	Kategorie für Bandverknüpfungen	
\$E	symbolisches Register für Barcode-Suche; wenn leer, wird der Barcode in folgenden Registern gesucht: ' : E', ' 8', ' 9 Z' (ab V2.29)	
\$F	Grund-Zeichengröße; von dieser werden alle anderen verwendeten Größen abgeleitet, die davon ein Vielfaches sind. Beispiele für G=22: Grundgröße: 22pt = G + 0 Klein-Schrift: 18pt = G - 4 Tabellen: 20pt = G - 2 Überschriften 26pt = G + 4 (Ab Modpar Version 3)	x
\$d	wenn leer oder = "r", Anzeige normal mit sho rec; wenn "b", Anzeige im Breitformat	x
\$H	Start-Text (name einer rtf-Datei ohne Erweiterung)	x
\$I	Liste der Ini-Flexe, die zu Beginn zu starten sind (mit Komma getrennt, letzter Eintrag darf ein Direkt-Flex (x ...\.) sein	x
\$S	Bibliotheks-Siegel	

\$u	Stamm-URL für Dateien (aus Web-Sicht)	
\$D	lokaler Stamm-Pfad für Dateien (aus Sicht des Dateisystems)	
\$U:	Update-URL	
\$R:	Update-Intervall (in Kalendertagen oder D W M)	
\$r:	Datum des nächsten Update-Checks	
\$Y:	Installationskennung (für Updates)	
\$y:	Codes für div. interne Steuerfunktionen	
	o:{f};	- Offline-Speicher an ({f}=1) oder aus ({f}=0)
	p:{n};	- Sleep nach put n ms
	l:{n};	- n=0: DB-Zugriff ohne Lock
\$a:	Codes für div. Anwendereinstellungen	
	F:{n};	n = Filenummer für Neusätze (für Trennung nach Arbeitsplatz)
	f:{f};	Formular-Mode: {f}=1 <Alt+E> beendet sofort {f}=0 <Alt+E> beendet n. letztem Formular
	s:{f};	Darstellung Sachgruppe {f}=1 Sachgruppe wird vor Signatur angezeigt {f}=0 Sachgruppe nicht anzeigen
	m:{c};	Medientypanzeige Code nicht gesetzt oder {c}=s Typ lt. Stammsatz {c}=a Typ lt. \$A-Schema
\$M	SMTP-Server	x
\$k	Name des Mailkontos	x
\$P	Passwort f. Mailkonto	x
\$o	-o -Parameter für SendEMail	x
\$e	Kennung Barcode-Generator (default: ESTD)	
\$z	Kennung Zugangsnummerngenerator (default: ZSTD)	
\$g	Kennung Signaturgenerator (default: (SSTD)	
\$l	Log-Level (default: alles loggen)	x
\$Z	Kategorie für ZS-Stammsatzverweise (default: #00)	
\$A	symbolische Bezeichnung des ALL-(Kreuz-)Registers	
\$T	Trennzeichen für Schnellsuche im ALL-Register default: <SPC>.,;/+<>='	

\$b nutzerspezifische Button-Funktionen: x
Format: x&x: Text={befehl}|Tooltip[@...]
\$v Proxy-URL
\$W PROXY-User
\$X Proxy-Passwort
\$f Zweigstellen-Kürzel der Zweigstelle - **nur privat!!**
\$O wenn belegt, Zweigstellenverwaltung aktiv

#uPI Prozess-ID

#uWP Working-Dir mit PID-Prefix, Inhalt z.B. c:\tmp\0123_

#uOp Operator wenn #op verfügbar = #op, sonst gleich username aus Environment

#uAc Access-Level zur Verwendung in Parameterdateien

#uKA Steuerung der Leser-Kontenanzeige (FlexLend)

#uÆI max. Anzahl der im Index gebildeten Schlüssel; dient dem Schutz vor Speicherüberlauf (Codierung OstWest!!!)

#uÆi Zähler für Indexschlüssel

\$MP_ProtectedVars

Liste der u-Variablen, die einen a99-Neustart "überleben" sollen. Enthalten sind die Variablen durch Komma getrennt mit führendem '#';

Beispiel: #uKA,#uXX, ...

6. Konventionen zur Programmierung

6.1. Variablen

temporäre Variablen: einbuchstabile \$-Var., überleben keine Unterprogramme!

#u-Variablen: wenn verwendet, werden sie mittels Push/Pop im UP gerettet und wiederhergestellt

6.2. Unterprogramme

Namen von UP's: **{modulkürzel}_Name** für über 'perform Sprungmarke' aufzurufende Unterprogramme

{modulkürzel}_Name() für über Stack aufrufbare Programme, die über 'jump Return' zurückkehren

6.3. Verzeichnisse

{PR} hier werden alle datenbank-unabhängigen Parameterdateien gesucht

{DB} - Datenbank-Dateien
- DB-spezifische Parameterdateien (*.ap?, *.cfg, *.ini)
- Log-Dateien (*.log, *.lst)
- Konfigurationsdateien für einzelne Module (*.lec, *.ini)
- Vorgabe-Listen (*.vw)

{BIN} Kommandozeilen-Hilfsprogramme, die von a99 über Systemaufrufe angesprochen werden

Es werden unter Allegro mitgeliefert:

winvi32 in Version 2: für Inspektion der Datenbank-Inhalte und zur Anzeige der Datenbank-Logdatei in Programm-Scripten

curl: Suche nach und Herunterladen von Updates von aneg-dv.de/allegro/updates
Hochladen von Dateien auf Verbundserver oder ähnliche Ziele

convert.exe: (ImageMagick): zum scriptgesteuerten Bearbeiten von z.B. cover-Dateien

sendEmail: wird zum Versand von E-Mails aus Allegro heraus genutzt

zip/unzip: zum Packen von internen Backups u. Auspacken von Update-Paketen

sounder: zum scriptgesteuerten Abspielen besonderer Töne bis hin zu Sprachansagen

ANEG-Teleservice: TeamViewer-Kundenmodul für den TeleService

7. Verfahren zur Programmaktualisierung

7.1. Aktualisierungswege

Programm(-Parameter)-Aktualisierungen werden auf zwei Wegen unterstützt:

- über eine Aktualisierungsdatei, die im Unterverzeichnis "updates" des Datenbank-Verzeichnisses manuell abgelegt wird
- über das automatische Herunterladen von Aktualisierungsdateien

7.2. allgemeine Konventionen zum Aktualisierungsverfahren

Die Aktualisierungsdatei ist ein zip-Archiv.

Name der Aktualisierungsdatei für das Update:

`upd_{modul}_{version}.zip`

Dabei sind:

{modul} das Modul-Kürzel, wie z.B.

mp - ModPar-Grundpaket

usr - anwenderspezifische Dateien

{version} eine Versionsangabe in der Form

MM_mm mit MM=Major- und mm=Minor-Version

In dem Archiv müssen zwingend enthalten sein:

`upd_install.flx` steuert die Details des Installationsprozesses

`upd_{modul}_{version}.rtf` enthält eine Beschreibung des Update-Inhaltes

Soll eine Datei gegen die Aktualisierung geschützt werden (weil z.B. lokale Anpassungen enthalten sind), kann diese mit der Kennung "@@usr" in einer der ersten drei Quelltextzeilen gekennzeichnet werden.

Automatische Benachrichtigung:

Ist im Feld #9A\$R des DB-Konfigurationssatzes (privat oder global) ein Update-Intervall eingetragen, wird zum Startzeitpunkt des Tages, der in #9A\$r enthalten ist, auf das Vorhandensein von Updates geprüft:

?		Update-Intervall gesetzt	_____
j		? Aktualisierungsdatum gesetzt	_____
		n update abrufen	_____
		j ? Frist abgelaufen	_____
		j update abrufen	_____
		Aktualisierungsdatum merken	_____

8. Funktionsbibliotheken

8.1. *MP_Misc.FLB (allgemeine Hilfsfunktionen)*

:MP_Log

IN: iV - Meldung: Beginn mit "!" - Warnung
 Beginn mit "@" - Fehler
 sonst Info-Eintrag
#uPI enthält Prozeß-Token
\$Modul - Modulname (Global vom aufrufenden Flex
gesetzt)
OUT: Eintrag in (dbn)_log.lst
 iV unverändert
DESC: Meldung aus iV wird in Log-Datei geschrieben
 Log-Datei wird dazu durch Umbenennen verriegelt

:a99_Restart

IN: iV - Startparameter für a99 (wird in _start.flx
ausgeführt)

DESC: in Datei 'emergency.rec' wird aktuelle
Recordnummer und beim Start auszuführender
Flexbefehl abgelegt
Beispiel: 1230045;x sho rec\dis
Verarbeitung dieser Informationen erfolgt in _start.flx:
dort wird Fkt. "a99_CheckRestart" aufgerufen

:a99_CheckRestart

IN: Datei 'emergency.rec' wird ausgewertet (s.

a99_Restart)
enthaltener Flex-Befehl wird extrahiert
OUT: iV ist leer, wenn kein Automatikstart;
iV = enthält "ok", wenn Automatikstart
oder den extrahierten Flexbefehl
FKT: lädt Record und startet evtl. Flex; löscht
emergency.rec
VAR: \$t, \$f

:MP_FileFind

Fkt: sucht Datei in Standardreihenfolge: DB, Flex,
Help, Prog, Start
IN: Dateiname vollständig ohne Pfad
Out: iV = volle FSP; "", wenn nicht gefunden
Var: \$t,\$x
\$MP_HELPSDir, \$MP_FLEXDir als Cache

:MP_FilesDel

Fkt: löscht alle enthaltenen Dateien
IN: iV enthält Liste der Dateinamen, mit cstring n
getrennt
OUT: iV leer

:MP_Var0Del

FKT: Löscht alle \$0-Variablen, die mit dem Inhalt der iV
beginnen
IN: \$0-Variablenname oder Anfang davon
OUT: iV unverändert
DESC: vor V28.2 muß mindestens 1 \$0-Var. übrigbleiben
(Dummy setzen!)

:MP_Var1Del

FKT: Löscht alle \$1-Variablen, die mit dem Inhalt der iV
beginnen

IN: \$1-Variablenname oder Anfang davon
OUT: iV unverändert
DESC: vor V28.2 muß mindestens 1 \$1-Var.
übrigbleiben (Dummy setzen!)

:MP_CompareEx

Fkt: prüft exakte alphanumerische Übereinstimmung
IN: #uCE 1. Wert
#uCe 2. Wert
OUT: iV = "", wenn ungleich
iV = #uCe, wenn gleich

:MP_VL_Init

FKT: Initialisiert neue Viewliste
DESC: name kann volle Pfadangabe enthalten
max. Länge der Gesamtliste auf 64000 Zeichen begrenzt
IN: iV "Name;Länge" der neuen Viewliste
OUT: iV "", wenn alles o.k.
!Fehlermeldung bei Fehler
VAR: \$AE_VL_Temp, \$AE_VL_Name, \$AE_VL_Len

:MP_VL_Add

FKT: Schreibt Zeile in Viewliste
DESC: max. Länge der Gesamtliste auf 64000 Zeichen
begrenzt
IN: iV = Zeile, die der akt. Viewliste hinzuzufügen ist
OUT: iV unverändert, wenn o.k.
iV leer, wenn Fehler; Fehlertext dann in \$MP_VL_Add
VAR: \$AE_VL_Temp, \$AE_VL_Name, \$AE_VL_Len
#uTm - temporäre Var.

:MP_VL_Write

FKT: Schreibt Viewliste aus Puffer, leert Puffer
IN: -

OUT: "", wenn o.k., sonst Fehlermeldung
DESC: wenn Fehler, werden Puffer nicht geleert
VAR: \$AE_VL_Temp, \$AE_VL_Name, \$AE_VL_Len

:MP_VL_Show

FKT: zeigt letzte Viewliste aus Puffer
IN: "" oder die drei Überschriften durch """" getrennt
OUT: \$vMP_VL_Show = "", wenn o.k., enthält sonst Fehlermeldung
sonst iV wie nach View
VAR: \$AE_VL_Temp, \$AE_VL_Name, \$AE_VL_Len

:MP_VL_Reset

IN: -
OUT: (iV gelöscht)
FKT: löscht Variablen und .vw?-Dateien
VAR: \$AE_VL_Temp, \$AE_VL_Name, \$AE_VL_Len

:MP_Increment

IN: iV - mit zu erhöhender Zahl n
OUT: iV mit n+1
FKT: erhöht Zähler in iV
VAR: \$mp_inc, mp_incz

:MP_Mkdir

FKT: Verzeichnis anlegen
IN: iV - Verzeichnispfad
OUT: iV = "", sonst Fehlertext mit "!"
eingeleitet
DESC: Prüft, ob Verzeichnis vorhanden; wenn nicht wird Verzeichnispfad komplett angelegt
Verzeichnis-Trenner: Backslash
verwendet 'open x' - es darf also keine Ausgabedatei offen sein!
VAR: \$t, \$td, \$mp_mkdir

:MP_EscBackslash

FKT: ersetzt \ durch \\ in iV
IN: iV
OUT: iV
VAR: #uU1, #uU2

:MP_SetSignal

FKT: setzt Signalfile (dbn.sgf)
IN: Info-Text für Signalfile
OUT: iV = "", wenn o.k, sonst Inhalt der Signaldatei
VAR: \$mp_setsignal
\$mp_setsignal_fn
\$mp_setsignal_tfn

:MP_ResetSignal

FKT: setzt Signalfile (dbn.sgf) zurück (wird gelöscht)
IN: -
OUT: iV = "", wenn o.k, sonst Fehlertext
VAR: \$mp_resetsignal_fn

:MP_FlipSet

FKT: setzt/löscht Flip-Text
IN: iv = {flipnummer}{FlipKurztext}={Flexbefehl}
{Fliplangtext}
Beispiel: 1&1: Daten=x sho rec|Satz im Auswahlfeld anzeigen
steht in der iV nur die Flipnummer, wird der Flip gelöscht
OUT: iV unverändert, wenn o.k, sonst "@Fehlertext"
VAR: \$par, \$flipArg, \$flipNummer - temporär
\$SYS_Fliptext1 ... \$SYS_Fliptext8 Aufbewahrung der Langtexte

:MP_Ver

FKT: ermittelt numerisch richtige A99-Versionsnummer mit 2 Stellen hinter dem Komma

IN: -

OUT: iV Versionsnummer in der Form MM.NN

:MP_GetTempFileName

FKT: bildet einen tmpörären Dateinamen (ohne extension)

IN: {vorsatz}

OUT: Name einer temp. Datei ohne Erweiterung im Temp-Verzeichnis

:MP_ListAdjust

Fkt: Liste von Doubletten bereinigen (Trennung durch Code 20dez)

IN: iV = Name der Liste

OUT: iV = "", wenn Liste bereinigt, sonst "!Fehler"

:MP_Trace

FKT: Schreibt Info in \$TraceLog

IN: iv-Inhalt

OUT: -

VAR: \$TraceLog, \$vTrace

:MP_IdCntInc()

FKT: liefert nächsten verknüpften Zähler für eine ID-Nummer

IN:

{register}<SPC>{begriff}<SPC>{tz}<SPC>{stellenzahl}

OUT: \$iV = nächster Zähler

= "", wenn Fehler; dann \$IdCntInc_Err belegt

DESC: Aufruf über Stack-Mechanismus (PushAdr/jump)

Nicht mehrplatz-sicher!

Der Zähler muss numerisch sein.

{stellenzahl} wird nur verwendet, wenn kein Zähler gefunden; ansonsten wird die Stellenzahl aus dem gefundenen Zähler abgeleitet

Beispiel:

für a0001

a0001*002

im Register 9

liefert der Aufruf mit "I9 a0001 * 2" den Wert 003

VAR: \$IdCntInc_Err

\$t temporär

\$p_IdCntInc_* interne Parameterspeicher

\$v_IdCntInc_* interne Zwischenergebnisse

#uDF - wenn gesetzt, erfolgen Debug-Meldungen

RINC: MP_Stack.flb

:MP_FetchEx

FKT: liest Datei ein bei w„hlbarer Ende-Zeichenkette

IN: iV = Codefolge, bis zu der gelesen werden soll

OUT: iV mit "", wenn nichts mehr zu lesen, sonst Text

VAR: \$vMPRE_Ende

\$vMPR_ECodes

\$vMP_FetchEx_Ret

DESC:Byte-Codes als Dezimalwerte mit Leerzeichen getrennt

:MP_Sound

FKT: spielt wav.Datei im Hintergrund

IN: name der wav.Datei ohne Extension

OUT: iV = "", wenn o.k., sonst mit ! eingeleitete

Meldung

VAR: \$vMPS - temp.

DESC:spielt wav-Date mit dem in der iV übergebenen

{namen}.wav

Die passende wav-Datei muss dazu im {BIN}-Ordner liegen

Beispiel: var "warn"\perf MP_Sound\if "!" mes

Benötigt sounder.exe im {BIN}-Verzeichnis

:MP_Message

FKT: wie message, aber mit Ton

IN: iV = Meldungstext

OUT: iV unverändert

VAR: \$vMP_Mes - temp.

DESC: zeigt mes-Box und spielt default.wav

8.2. MP_ACC.FLB (Berechtigungs-Funktionen)

:MP_AccCheck()

FKT: Prüft Zugriffsberechtigung

DESC: nutzt MP_Call-Mechanismus

IN: iV = erforderliche Mindest-Berechtigungsstufe

OUT: \$iv = "", wenn o.k., ansonsten wird

access_denied-Meldung

ausgegeben und \$iv enthält Inhalt von cstring A

RINC: mp_uif.flb, mp_stack.flb

8.3. MP_DaTim.FLB (Datums- & Zeitberechnungen)

:MP_Date2Sort

FKT: wandelt Datumsformat von [T]T.[M]M.[JJ]JJ in JJJJMMTT

IN: iV - zu wandelnde Zeichenkette

OUT: iV - umgewandelte Zeichenkette

leer, Wenn Fehler (ausserhalb 01.01.1000 bis 31.12.9999)

2-stell. Jahreszahl wird in Bereich zw. 1980 und 2079 transform.

VAR: #uÆ0,#uÆ1,#uÆz veraendert

DESC: Das Datum wird formal gwandelt; es erfolgt keine Prüfung auf Zulässigkeit des Tags im Monat

:MP_Sort2Date

FKT: wandelt Datumsformat von JJJJMMTT in [T]T.[M]M.[JJ]JJ

IN: iV - zu wandelnde Zeichenkette

OUT: iV - umgewandelte Zeichenkette

leer, Wenn Fehler

VAR: #uÆ1

:MP_IsDate

Fkt: Datumsangaben normalisieren

IN: in iV wird folgende Datums-Form erwartet

[t]t.[m]m.jjjj

[t]t.-[t]t.[m]m.jjjj

[t]t.[m]m.-[t]t.[m]m.jjjj

OUT: iV wird normalisiert in

[t]t.[m]m.jjjj

oder
[t]t.[m]m.-tt.mm.jjjj

wenn die Eingangszeichenkette nicht dem Schema entspricht, wird "" zurückgegeben

Var: #ud1, #ud2
\$tag, \$monat, \$jahr
\$etag, \$emonat, \$ejahr
\$isdate, \$isDate_z

:MP_TimeDiff

FKT: ermittelt Zeitdifferenz
IN: iv: zwei Zeitangaben hh:mm:ss mit "-" getrennt
OUT: Differenz in hh:mm:ss
VAR: \$Æ_TD, \$Æ_TDZ, \$_aeTDz

:MP_TimeDiffSec

FKT: ermittelt Zeitdifferenz in Sekunden
IN: iv: zwei Zeitangaben hh:mm:ss mit ";" getrennt
OUT: Differenz in Sekunden

:MP_Time2Sec

FKT: liefert Zeit in Sekunden
IN: iv: Zeitangabe in hh:mm:ss
OUT: Wert in Sekunden
VAR: \$mp_T2S

:MP_Sec2Time

FKT: liefert Zeit in hh:mm:ss
IN: iv: Zeitangabe in Sekunden
OUT: Zeit in hh:mm:ss

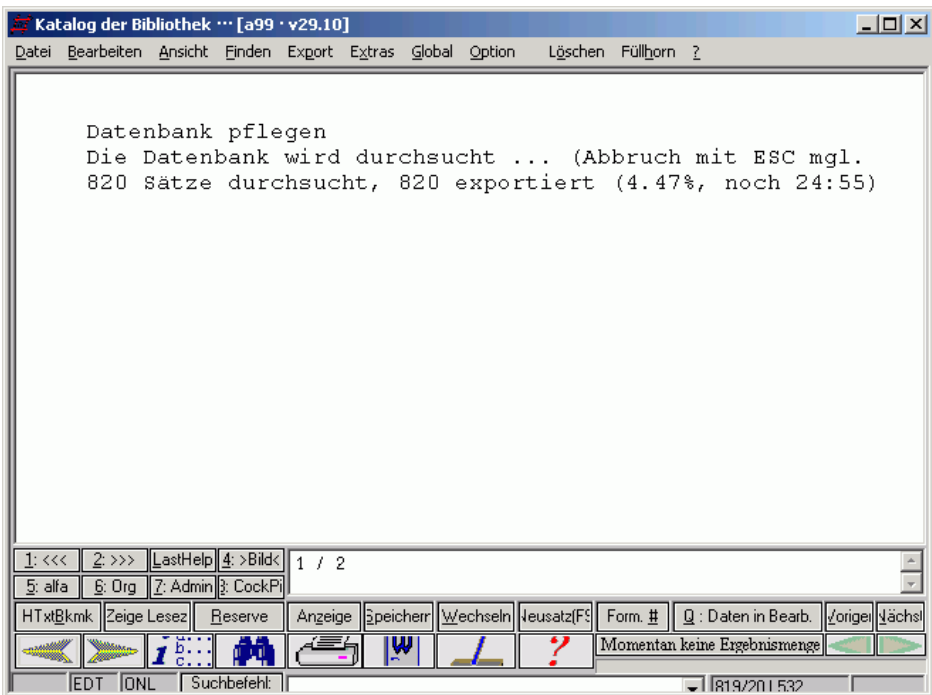
:MP_PercRemain

FKT: prozentualer Restanteil
IN: iv: n-Max;nAkt
OUT: prozentualer Restwert; Format: nn.nn%

:MP_TimeRemain

FKT: Restzeit eines Prozesses
IN: iv: Startzeit;akt_Zeit;max;akt
OUT: Restzeit in [[hh:]mm:]ss; wenn unbestimmt, dann
"?"

Die Zeitberechnungsfunktionen ermöglichen informative Anzeigen wie
in diesem Beispiel:



8.4. MP_Stack.flb (Stackfunktionen)

:PushRec

Fkt: legt akt. Satznummer auf Stack
(\$MP_RecordStack)
VAR: \$MP_RecordStack, \$vPushRec
OUT: akt. Rec-Nr. im Stack gespeichert

:PopRec

Fkt: lädt Satz mit Satznummer vom Stack (#uRS)
VAR: #uRS, #uÆ0
OUT: neuer Rec geladen; wenn Fehler, \$vMP_Err
besetzt

:DropRec

Fkt: entfernt Satznummer vom Stack
(\$MP_RecordStack)
VAR: \$MP_RecordStack, \$vDropRec
OUT: \$vMP_Err besetzt

:PushAdr

FKT: Merkt Rücksprungmarke in AdressStack
max. Stackgröße ca. 31940 Bytes (a99 fliegt bei
Variablen, die länger als 32000 Bytes werden
IN: iV - Rücksprung-Marke
OUT: iV leer oder Fehlermeldung, wenn Stacküberlauf
DESC: PushAdr / Return unterstützen einen
schachtelbaren UP-Mechanismus, der über den jump-
Befehl funktioniert:
- vor UP-Aufruf wird mit 'perform MP_PuschAdr' die
Rücksprungmarke im Stack abgelegt
- das UP wird mittels 'jump UP-Name' "aufgerufen"
- Das UP endet mit 'jump Return'

- der Markenname darf nicht länger als 60 Zeichen sein und kein Semikolon, code 9 (TAB) oder " " (Space) enthalten

Beispiel:

```
-----  
var "M"\perf PushAdr          // Rücksprung-Marke auf  
                               // Stack  
var "par1"\perf Push         // Parameter werden über  
                               // den Stack übergeben  
  
var "Par2"\perf Push  
jump UP                       // ruft UP auf  
:M                             // hierher kehrt das UP  
                               // zurück  
var $iv\meslend              // gibt "hab' was getan"  
                               // aus  
  
-----  
:UP  
perf Pop\ins $p2              // Parameter rückwärts  
perf Pop\ins $p1              // wieder aus Stack holen  
var "hab' was getan"  
jump Return                   // Rücksprung auf Marke  
                               // aus Stack  
  
-----
```

VAR: \$MP_AdrStack, \$MP_AdrSP, \$t

:Return

FKT: Springt auf letzte Marke aus Stack

IN: iv - Rückgabewert der Funktion bzw. des
Unterprogrammes

OUT: iv zerstört

\$iv enthält geretteten iv-Inhalt

DESC: mit jump zu nutzen, nicht mit Perform!!!

s.a. PushAdr

VAR: \$MP_AdrStack, \$MP_AdrSP, \$iv, \$t, \$mp_retadr

:Push

IN: iV
OUT: iV unverändert
FKT: iV-Inhalt wird auf Stack gelegt
DESC: Stack wird in \$-Variablen abgelegt; max. Länge ist 999
Eintraege
VAR: \$MP_ivStacknnn - Stack-Variablen
\$MP_ivSP - aktuelle Stacktiefe
\$vPush - temp.
\$vPushz - Zwischenspeicher für z
\$vPushZ - dto. für Z

:Pop

IN: -
OUT: iV aus Stack aktualisiert
FKT: holt naechsten Wert vom Stack
VAR: \$MP_ivStacknnn - Stack-Variablen
\$MP_ivSP - aktuelle Stacktiefe
\$vPop - temp.
\$vPopz - Zwischenspeicher fuer z

8.5. MP_DB.FLB (Datenbank-Funktionen)

:MP_CfgGetValue

Fkt: Liefert Wert einer cfg-Zeile
Desc: liest Zeile, die mit Inhalt der iV beginnt
IN: iV = cfg-Kennung
OUT: Wert/Inhalt des adressierten Wertes oder ""
wenn mit ! beginnt, Fehlermeldung
RINC: MP_Misc.FLB

:MP_ChkReg

IN: iV enthält Wert in der Form wert@register@truncate
wert ist der zu suchende Begriff
register ist |3 o.ä. oder der symb. Registername
truncate wenn =1, dann darf der Registerwert länger
sein
OUT: iV = "", wenn Wert nicht gefunden, sonst Registerwert
Fkt: prüft das Vorkommen eines Wertes im Register
Var: #uRw - wert
#uRn - Register
#uRf - Truncate-Flag
#uRx/#uRX- temp.

:MP_Choose()

Fkt: wie choose, aber mit sofortiger Satzanzeige
IN: Suchbegriff oder '*' (wie choose)
OUT: wie choose; wenn iv leer, steht evtl. in #uÆE eine
Fehlermeldung
Satz ist jedoch im Gegensatz zu choose geladen
Desc: Aufruf über Stack-Mechanismus
(var 'rueckkehr"\perf PushAdr\...\)
jump MP_Choose():rueckkehr
RINC: MP_Misc.flb, MP_Stack.flb

:MP_Rec2EM

Fkt: fügt aktuellen Record einer benannten Ergebnismenge hinzu
IN: {Name}
OUT: iv = "", wenn kein Fehler; sonst mit !Fehlermeldung
Desc: Fügt den akt. Record einer Ergebnismenge mit dem Namen {Name} hinzu

:MP_CloseActRes

Fkt: löscht die letzte EM und aktiviert vorige
IN: -
OUT: Name der akt. Ergebnismenge, wie im EM-Fenster angezeigt
Desc: war die aktuelle EM nicht die letzte bleibt sie aktiv
EM mit Nummern <3 werden nicht aktiviert

:MP_RegisterCount

Fkt: zählt das Vorkommen eines Wertes im Register (A99-Version)
IN: iV enthält Wert in der Form wert@register@truncate
wert ist der zu suchende Begriff
register ist |3 o.ä. oder der symb. Registername
truncate wenn =1, dann darf der Registerwert länger sein
OUT: iV = Anzahl der zutreffenden Eintreage
Var: \$wert - wert
\$register - Register
\$fTruncate - Truncate-Flag
\$temp/\$tempWertEx - temporär

:MP_FndRecLock

Fkt: lädt und verriegelt Satz
IN: iV - Satznummer
OUT: Datensatz geladen und verriegelt
iV = "", wenn o.k., sonst Fehlermeldung: load, lock, new,
del, diff, inkonsistente Daten

:MP_Lock

Fkt: verriegelt Satz, lädt ihn neu und prüft
Übereinstimmung
IN: beliebiger Datensatz geladen
OUT: Datensatz verriegelt
iV = "", wenn o.k., sonst Fehlermeldung: lock, new, del,
diff, inkonsistente Daten

:MP_PutUnlock

Fkt: speichert und entriegelt aktuellen Satz
IN: beliebiger Datensatz geladen, der Neu, bearbeitet
und oder verriegelt sein kann
OUT: Datensatz gespeichert und entriegelt
iV = "", wenn o.k.
iV mit Meldung, wenn Fehler aufgetreten
DESC: im #0sf wird Modulinformation und Prozess-ID
abgelegt.
WORKAROUND put: mit set Err def. Flag-Zustand setzen
WORKAROUND set obj/offline-Datei wird immer gelöscht

:MP_FormLoad

Fkt: Lädt satzartspezifisches Formular
IN: iV = {Satzart-Kennung};[{Hilfekategorien}]
OUT: iV = "", wenn o.k, sonst !{Fehlermeldung}
DESC: löst Prompt-Vorgaben für Teilfelder auf, lädt
satzartspezifische Hilfedateien

mehrere Hilfekategorien werden durch Komma getrennt angegeben

Beispiel: var "TIT;2,4,6" lädt TIT.FRM und H2, H4 und H6

Var: \$vMP_FormLoad, \$vMP_FormLoad_Kat,
\$vMP_FormLoad_Typ, \$t, \$p, \$s

:MP_FormRelease

Fkt: Lädt datenbankspezifisches Formular und löscht temp.

Hilfedateien he_ad und H{Hilfekategorie}

IN: [{Hilfekategorien}]

OUT: iV = "", wenn o.k, sonst !{Fehlermeldung}

Var: \$vMP_FormRelease, \$t

:MP_FormProc

IN: -

OUT: iV = "", wenn o.k, sonst !{Fehlermeldung}

Flags wie nach Befehl form (can)

Fkt: ruft alle Formulare der aktuell geladenen
Formulardatei auf

DESC: benutzt open - evtl. vorher offene Input-Datei
schließen!

Var: \$vMP_FormProc, \$t, \$t1, \$vMP_FLT

:MP_QTExpand()

FKT: Erweiterung eines Suchbegriff-Terms über
Verweise

IN: iV - Suchausdruck in der Form REG SRCH

REG - symbolischer Registername oder
Direktangabe

SRCH - Suchausdruck

OUT: iV - umgewandelter Suchausdruck

beginnt mit '!', wenn Fehler und enthält dann Fehlertext

DESC: wandelt REG SRCH in
(REG SRCH [OR REG RSRCH1 [OR REG RSRCH2
[OR REG RSRCHn]]]) um
Dazu werden Stammsatzverweisungen im gleichen
Register genutzt
als Verweiszeichen wird " -> " genutzt
Ein Suchbegriff muss mindestens 2 Zeichen lang sein
(ohne '?')
Aufruf über MP_Call-Mechanismus!
RINC: MP_Misc.flb

:MP_ResultSort

Fkt: sortiert Ergebnismenge nach dem Inhalt einer
Kategorie.
IN: bestehende Ergebnismenge
iV = Kategorienummer und Sortierrichtung in der Form
#nnn|s
s = Sortierrichtung: a - aufsteigend, d =
absteigend

OUT: iV = "", wenn o.k., sonst !{Fehlermeldung}
neue aktuelle Ergebnismenge
Desc: Nichtsortierworte werden entfernt, Index-
Sortieraufbereitung wird verwendet (p-Tabelle)
VAR: \$fMPRS_Des - besetzt, wenn absteigend

:MP_FindFirst

Fkt: lädt ersten Satz in Registerreihenfolge
IN: {register}<SPC>{begriff}
OUT: iV = "", wenn nichts gefunden
sonst #{recordnummer} und Flags wie nach find #
Desc: In {register} wird nach {begriff} gesucht. Wird ein
Eintrag
mit übereinstimmendem Anfang gefunden, wird der erste
Satz mit find # geladen

8.6. MP_Dis.FLB (Anzeigefunktionen)

:MP_Dis

FKT: aktualisiert Satz-Anzeige
IN: -
OUT: iV unverändert
DESC: Bei der Aktualisierung der Anzeige wird die Einstellung aus dem Datenbank-Konfigurationssatz berücksichtigt

:MP_DSet

FKT: setzt Anzeigebreite
IN: -
OUT: iV unverändert

:MP_DisClear

FKT: löscht Anzeigefeld
IN: -
OUT: iV unverändert oder enthält "ÆÆÆ" (CP850), wenn Abbruch gewünscht
VAR: \$AE_Display

:MP_DisAdd

FKT: aktualisiert Anzeigefeld
IN: iV: an Display anzuhängenden String (rtf)
OUT: iV unverändert oder iV enthält "ÆÆÆ" (CP850), wenn Abbruch gewünscht
VAR: \$AE_Display

:MP_DisAddLine

FKT: aktualisiert Anzeigefeld
IN: iV: an Display anzuhängenden Zeile (ascii); "\par" wird

angehängt
OUT: iV unverändert
VAR: \$AE_Display

:MP_DisAddLineHidden

IN: iV: an Display anzuhängenden Zeile (ascii); "\par
" wird
angehängt
OUT: iV konvertiert
FKT: aktualisiert Anzeigepuffer ohne Refresh
VAR: \$MP_Display

:MP_DisShow

FKT: aktualisiert Anzeigefeld, iv-Inhalt wird nicht
gemerkt
IN: iV: an Display anzuhängenden String (rtf)
OUT: iV unverändert
VAR: \$AE_Display

:MP_SfMes

FKT: Meldung im Schreibfeld ausgeben, auf
Bestätigung mit Strg+Y warten (Schreibfeldinhalt
wird im Abschluss wiederhergestellt)
IN: iV = Meldungstext in ascii
OUT: iV leer
VAR: \$t, \$tsf, #uÆt

:MP_UifTxtGet

Fkt: liest UIF-Text aus UIF-Datei
Aufruf ueber MP_Call-Mechanismus
IN: Stack: Code (einschließlich "UIF", z.B. "UIFe")
Textnummer
OUT: UIF-Text in ansi oder leer, wenn Fehler
RINC: MP_Stack.flb
VAR: \$MPUIF... - Zwischenspeicher fuer UIF-Datei
\$tnr - IN: Textnummer
\$code - IN: UIF-Dateicode
\$resttext - temp. noch nicht vergl. UIF-Inhalt
#uli - akt. bearbeitete UIF-Zeile
\$t - temp. Var.
DESC: - ermittelt UIF-Text anhand der Sprachversion
- UIF-Datei wird in \$-Var zwischengespeichert

8.7. MP_FuncList.FLB (dynamische Menü's)

:MP_FuncList()

FKT: bietet Liste von Funktionen zur Auswahl an, die als Flex-Datei im DB-Verzeichnis abgelegt sind.

In Flex-Dateinamen erlaubte Zeichen: a-z A-Z _ - .

IN: iV - [(kürzel)];(Überschrift)]

(kürzel) - Teilstring, der in den Flex-Dateinamen gesucht wird

(Überschrift) - wird im Listenkopf angezeigt

#u/Es: wenn besetzt, keine Sofortauslösung von Flexen

DESC: Aufruf über Stack

Es werden folgende Szenarien für die Funktionsauswahl über dynamische Menüs unterstützt:

- A) alle Funktionen, deren Aktivierungsbedingung erfüllt ist, werden angeboten
- B) alle Funktionen wie A) werden angeboten, die außerdem im Dateinamen den Teilstring (kürzel) enthalten

Der Inhalt der Flexdatei wird zeilenweise ausgewertet, bis eine Leerzeile oder die Zeichenkombination @@ auftritt
Es werden alle Zeilen interpretiert, die einen String XXX: enthalten:

XXX: Bedeutung/Syntax

CHR: Char-Set, in dem die Beschreibungen und Texte codiert sind
Werte: ansi oder ascii
default: ascii

FKT: Zeile enthält die Bezeichnung der Funktion so, wie diese in der angebotenen Auswahl angezeigt werden soll

REQ: Bedingungen, die für die Aufnahme der Funktion in die

Auswahl erfüllt sein müssen. Eine Bedingung kann wie folgt definiert werden:

`[!]{(feld)}${(teilstreng)}[(op)(teilstreng)]{(if-bedingung)}`

(feld) Kategorie oder Variable, z.B: #20, #u11, \$xyz
(teilstreng) Teilfeldzeichen
(op) Operator: = - beginnt mit
oder % - enthält
(teilstreng) Zeichenkette, die in der Kategorie bzw. in deren Teilfeld gesucht werden soll; darf keine Leerzeichen enthalten
(if-bedingung) unterstützt wird: main, New, hasAuf[(@{registernummer}){register}], hasVol[(@{registernummer}){register}]

Mehrere Bedingungen werden durch ';' (AND) oder '|' (OR) getrennt.

Die Funktion wird nur aufgelistet, wenn die definierte(n) Bedingung(en) durch den aktuell geladenen Satz erfüllt ist/sind.

Rang der Operatoren (aufsteigend): |;!=%

Leerzeichen sind im Ausdruck nicht erlaubt, Klammerungen werden nicht unterstützt.

Beispiele:

#9B muß vorhanden sein und mit 0 oder 1 beginnen:

REQ: #9B=0|#9B=1

#9B muß vorhanden sein, darf aber nicht mit 2 beginnen:

REQ: #9B;!#9B=2

Im Satz solle #00 und #20 oder #00 und #19 enthalten sein:

REQ: #20;#00|#19;#00

Test auf verknüpft gespeicherten Aufsatz (* in #00):

REQ: #00%*

Einschränkung auf Suchkürzel: Da ohne (kürzel) alle Flexe aktiviert werden, deren Bedingung erfüllt ist, werden u.U. zu viele Funktionen angeboten.

Damit ein Funktionsflex nur angeboten wird, wenn ein (kürzel) als Parameter angegeben wurde, kann mittels REQ: #uÆp=(kürzel) genauer eingegrenzt werden.

Beispiel:

REQ: #uÆp=new | wahr, wenn als Such(kürzel)
| 'new' angegeben wurde

HELP: (rtf-Datei)|(html-Datei)

für Anzeige: rtf-Datei (ohne Endung angeben)

für aresqua: html-Datei (mit Endung)

Beispiel: HELP: Name.html

oder HELP: Name

PAR: Parameter, der der Funktions-Flexdatei übergeben werden soll

SRT: Wert, nach dem die Einträge der fertigen Liste sortiert werden (0 ... 65535); Fehlt SRT: gilt SRT: 0.

MNU: 0 : Kontextmenü verbieten, wenn nur eine Auswahl mgl.
1 : Kontextmenü erzwingen, wenn nur eine Auswahl mgl.
(default: wie vorher)

ACC: minimaler Berechtigungs-Level

Kommentare: alles ab "/" wird in einer Zeile ignoriert. Beginnt eine Zeile mit "/", so wirkt dies wie eine Leerzeile und beendet die Auswertung der aktuellen Flexdatei

vor a99, V27.2:

In Anzeigeparametern muß Abschnitt #-i (EC hex) enthalten sein, der die Umwandlung der Var. #uÆp in Kleinschreibung realisiert:

```
;---- Abschnitt zur Umcodierung, verwendet in Æ_FuncList ----  
#-i  
!uÆp y2 dÆp aÆp  
#+#  
;-----
```

RINC: MP_Stack.flb

OUT: (ausgewählte Funktion wird ausgeführt)

VAR: #uÆp - Zwischenspeicher für Parameter (kürzel)

\$vFIFn - dto. für Flex-Name

\$vFISw - dto. für Sortierwert

\$vFIKi - Zwischenspeicher für notw. Kat-Inhalt

\$vFIFb - Funktionsbezeichnung aus Flex-Datei

\$vFLMnu - Menü-Flag (s. MNU:)

#uÆf - erforderliche Kategorie[Teilfeld]

#uÆh - Zw.Sp. Hilfezeile

#uÆt - Zw.Sp. Funktionszeile

#uÆa - minimale Berechtigungsstufe

#uÆu - Überschrift über Menü

#uÆv - minimale a99-Version

z u. Z werden verändert

\$fFound - Flag: mind. eine Datei gefunden

\$ÆFuncList - Liste der Flex-Dateien (Cache)

\$ÆFuncList - dto. zur Abarbeitung intern

\$OnKontext - wird gelöscht, wenn Liste mit ESC
verlassen

\$MenuStack - enthält übergeordn. Menü oder leer
(mp-fkt-Parameter)

DEBUG: wenn \$FuncList_DEBUG besetzt, Herkunft der
Menüpunkte im Menü

wenn #uDF besetzt, ausführliche Debug-Meldungen

:MP_MenuShowLast

Fkt: ruft letztes Menü, wenn vorhanden

IN: \$MenuStack

OUT: iV = "", wenn kein Menü in \$MenuStack

VAR: \$MP_MenuLast - enthält Kürzel für letztes Menü

DESC: Wenn Eintrag im Menü-Stack, wird letztes Menü gestartet

:MP_MenuStackClear

Fkt: löscht Menü-Stack

8.8. MP_Res.FLB (Ergebnismengen-Bearbeitung)

:MP_ResSortEx()

Fkt: sortiert eine Ergebnismenge
IN: aktuelle Ergebnismenge
iV - {Sortierparameter} [":r"]
OUT: iV = "", wenn Sortierung o.k., sonst "!" {fehlermeldung}
DESC: Aus der Ergebnismenge wird im temp-Verzeichnis eine Sortierdatei erzeugt, mit asort sortiert und als aktuelle Ergebnismenge wieder bereitgestellt
Die Sortierdatei erstellt je Satz eine Text-Zeile mit dem Sortierwert und der Satznummer, getrennt durch '|':

Beispiel: TexttextText|123456

Wenn nach dem Dateinamen ":r" angegeben ist, wird absteigend sortiert
Aufruf über Stack

Var: #uDF - wenn gesetzt, werden Debug-Infos ausgegeben
\$vMP_ResSortEx
\$vMP_RSE_dir - Sortierrichtung
\$vMP_RSE_sPara - Sortierparameter
\$vMP_RSE_2sFile - Sortierdatei

RINC: MP_Stack.flb
MP_Misc.flb

9. Hilfsroutinen und -programme

9.1. Cover-Verlinkung

MP_JanasShowCover.flx (noch in Entwicklung)

Aufruf: URL einer Bilddatei

Funktion: erzeugt eine temporäre html-Datei mit img-Tag zur Anzeige einer Grafik über Janas

9.2. URL-Darstellung und Verlinkung

MP_WinStart.flx

Aufruf: mit URL oder Pfadangabe einer Ressource

Funktion: bereitet Parameter auf und übergibt ihn für den Befehl "start" passend formatiert

Wenn "&"-Zeichen im Parameter vorkommen, wird für die Darstellung des Link-Ziels verwendet, da sich das &-Zeichen nicht über die Betriebssystem-Befehlszeile übergeben läßt.

9.3. Verwendung externer Hilfsprogramme

allegro (a99) nutzt für viele Aktionen Hilfsprogramme, die es über cmd aufruft und deren Ergebnisse dann von a99 weiterverarbeitet werden. Diese Hilfsprogramme werden entweder im Allegro-Stammverzeichnis (dort wo sich a99.exe befindet, {PR}) oder in dessen Unterverzeichnis {BIN} gesucht - das betrifft:

- in {BIN}: **zip.exe/unzip.exe** zip-tools von Infozip, nötig zur Installation von Updates und zum Packen/Entpacken von Export- und Importdateien
- in {BIN}: **curl.exe** für Internet-Zugriffe , z.B. für die direkte

Übernahme von Stamm- und Mediendaten von der DNB und anderen Datenquellen

(a99 kann zwar selbst auch TCP/IP-Verbindungen herstellen, ist jedoch nicht proxy-fähig)

- in {BIN}: **sendmail.exe** oder sendemail.* für den automatischen Versand von E-Mails aus Allegro heraus
- in {PR}: **zc.exe** für z29.50-Verbindungen (Datenabruf) zu anderen Bibliotheksservern
- in {BIN}: **winvi32.exe**; wird genutzt für die Anzeige der Anwendungs- und Datenbank-Logdateien sowie zur "Begutachtung" von Exportergebnissen oder zu importierenden Dateien.

10. Integration anwenderspezifischer Anpassungen

10.1. Zweck der Integration anwenderspezifischer Anpassungen

Abhängig von den beim Anwender geltenden Geschäftsregeln werden Inhalte wie Medientypen, Nummerngeneratoren, Erwerbungsarten und ähnliches verschieden verwendet.

Um anwenderspezifische Funktionen und Regeln in den Standard-Scripten zu berücksichtigen, werden spezielle Flex-Dateien mittels include eingebunden.

10.2. Anpassung der Exemplarerfassung

tit-adex.flx	vor Formularaufruf nach dem Formularaufruf	usr-tit-adex.inc usr_tit-adex_check.inc
exs-edt.flx	vor Formularaufruf nach dem Formularaufruf	usr_exs.inc usr_exs_aftr.inc
zsh-neue.flx	Voreinstellungen der autom. Exemplaranlage	usr-zsh-neue.inc

10.3. Anpassung der Statistik

mp-stat.flx	Statistik nach Zugangsjahr	mp-stat.inc
mp-stat-invent.flx	Bestandsstatistik	
	- nach Signaturgruppen	stat-sig.inc
	- nach Standort	stat-ort.inc
	- nach Sonderstandort	stat-sst.inc

Anhang A: Installation der "Modularen Parameter"

Vorbereitung der Datenbank:

- Ausführen des Installationsprogrammes, dabei werden
 - * Parameter-,
 - * Flex- und
 - * Hilfsdateienin das ausgewählte DB-Verzeichnis kopiert

- Anlegen des Datenbank-Konfigurationssatzes
 - * Einstellen der verschiedenen Parameter
 - * Einlesen der Standard-Satzartendefinitionen (mp-crt.alg)

Literaturangaben, Verweise

- [1] Die Idee, das temporäre als auch das Arbeitsverzeichnis unterhalb des Datenbankverzeichnisses anzulegen, wurde erstmals von Heinrich Allers in der Allegro-Liste veröffentlicht.

Stichwortverzeichnis

Anpassung.....	67
Anwender-Schaltfläche.....	20
Bedingungen.....	6
Button.....	20
Datenbank.....	6, 10, 12, 19, 24, 68f.
Datenbank-Konfigurationssatz.....	55
Datenbankverzeichnis.....	6, 19, 69
Dynamische Menüs.....	58
E-Mail.....	66
Ergebnismenge.....	10, 64
Exemplaranlage.....	67
Generator.....	25, 27
Generatorsatz.....	25
Hilfdatei.....	6, 13, 19, 51f.
Hilfsprogramme.....	29, 65
Indexdatei.....	19
Initialisierung.....	11
Initialisierungen (zusätzlich).....	12
Installation.....	27, 68
Konfigurationsdaten.....	24
Konfigurationssatz.....	19, 24, 26, 68
Menüs.....	5, 10ff.
Menüsteuerung.....	10f.
Modularisierung.....	5
MP_Stack.....	7, 10, 45
Neuerfassung.....	6
Neusatz.....	6, 18
Nummerngenerator.....	27, 67
Operator.....	19, 28
Programmstart.....	11
Programmverzeichnis.....	6
Quittungsdruck.....	24
Satzart.....	5, 10, 21, 23, 51, 68
Signatur.....	27
Sortierdatei.....	64

Sortierparameter.....	12, 64
Sortierrichtung.....	64
Stackfunktionen.....	10
Statistik.....	67
TCP/IP-Verbindungen.....	66
Update.....	5, 27
Vereinbarungen.....	6
Zugangsnummer.....	27