masc-ato

Automated Transaction Operator
User's Guide
***
VSE/MVS
Version 4.1.0

MATO-UG410-1-D

Bezugsquelle:	***masc ag*** Abteilung SWD Birkenstr. 49 CH-6343 Rotkreuz (Schweiz)
Telefon:	041 / 790 53 44 International: (+41) 41 790 53 44
Telefax:	041 / 790 53 40 International: (+41) 41 790 53 40
Bürozeiten:	8 - 12h, 14 - 17h MEZ (Mo - Fr)

Ausgabe März 1999.

Weitere Exemplare dieser Dokumentation können Sie mit beiliegendem Bestellformular anfordern.

VORWORT

Dieses Handbuch beschreibt die Automated Transaction Operator (ATO) Befehle und deren Anwendung. Es enthält weiter Anhänge mit

- **Anhang A: Beispiele von ATO Dialogen**
- **Anhang B: Migrations-Hinweise für Anwender der Vorversion**
- **Anhang C: Erweiterungen Version 4.1**
- **Anhang D: User Exit ATOEXI**

Änderungen zu dieser Publikation sind unter der Rubrik "Änderungsübersicht" zusammengefasst.

Leser dieser Publikation sollten über grundlegende Kenntnisse der VSE- resp. MVS-Funktionen sowie Erfahrung mit den TP-Monitor Bedieneroberflächen der entsprechenden Anwendungen verfügen.

Kenntnisse von REXX oder einer anderen Programmiersprache sind ebenfalls notwendig.

Übersicht der ATO-Dokumentation

- `MATO-HO410-1-D` ***masc-ato*** "Automated Transaction Operator": *Handout*
- `MATO-GI410-1-D` ***masc-ato*** "Automated Transaction Operator": *General Information*
- `MATO-UG410-1-D` ***masc-ato*** "Automated Transaction Operator": *User's Guide*
- `MATO-IN410-1-D` ***masc-ato*** "Automated Transaction Operator": *Installation Guide*
- `MATO-MC410-1-D` ***masc-ato*** "Automated Transaction Operator": *Messages and Codes*
- `MATO-SA410-1-D` ***masc-ato*** "Automated Transaction Operator": *Samples*

ÄNDERUNGSÜBERSICHT

Dies ist die erste Ausgabe von ***masc-ato*** "Automated Transaction Operator": *User's Guide*

Im Anhang ist eine Übersicht des erweiterten ATO-Befehlssatzes und die Migration von bestehenden ATO-Dialogen beschrieben.

Erweiterungen

masc-ato wurde komplett überarbeitet und mit weiteren Befehlen und Parameter ergänzt. Die neuen Befehls-Optionen wurden derart gestaltet, dass sie in vielen Teilen logisch untereinander zusammenspielen. Grosses Augenmerk wurde darauf gelegt, dass die Befehle übersichtlich und gut lesbar sind.

Wir danken allen Anwendern, welche mit hilfreichen Anregungen mitgeholfen haben, die neuen Erweiterungen zu gestalten.

CICS: Die in dieser Dokumentation aufgeführten Beispiele basieren auf CICS/ESA.

Änderungen Ausgabe März 1998

Neuer ATO_SET_SESSION_PARAMETER Parameter LOGAID zeigt auf dem Logterm die Taste, die gedrückt wurde.

Neue Prolog Parameter:
- ☞ LOGAID zeigt auf dem Logterm die Taste, die gedrückt wurde
- ☞ MLOG schreibt alle durchlaufenen Makros auf ATOPRT
- ☞ ASMGEN und ASMDATA steuern den Output des Assembly Steps

PUTPRT akzeptiert neu ein Label, das angesprungen werden kann.

Die Default-Werte der LIN/ENDLIN und COL/ENDCOL Parameter im SCAN Macro ändern aufgrund der tatsächlichen Eingaben.

Änderungen Ausgabe März 1999

- ☞ @b und @f statt @t
- ☞ ESC Parameter bei ATO_SET_SESSION_PARAMETERS

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

1. NOTATIONS-KONVENTIONEN

Das Format der ATO-Befehle wird durch die folgenden Symbole ergänzt:

- Eckige Klammern [] spezifizieren optionale Felder oder Parameter.
- Ein vertikaler Balken | unterteilt eine Auswahl. Falls nicht anders erwähnt, kann nur eine Alternative ausgewählt werden.
- **FETTDRUCK**-Buchstaben müssen wie beschrieben eingegeben werden.
- *Kursiv*-Buchstaben bezeichnen Felder oder Parameter, die einem Befehl mitgegeben werden müssen.
- Anwendungsspezifische Zusätze:

o	**VSE**	Operating System
o	**MVS**	Operating System
o	**VM/CMS**	Operating System
o	**VTAM**	Netzwerke
o	**CICS**	TP-Monitor
o	**IMS**	TP-Monitor
o	**TSO**	TP-Monitor
o	**SAP**	Anwendungs-Software
o	**COPICS**	Anwendungs-Software
o	**MRPS**	Anwendungs-Software

2. ALLGEMEINE NOTATIONS-REGELN

2.1. Allgemeines

Die Notations-Regeln von ***masc-ato*** sind formfrei und richten sich nach der jeweiligen Programmiersprache.

2.2. Kompatibilitäts-Modus

Die Notations-Regeln des Kompatibilitäts-Modus richten sich nach den Assembler-Konventionen. Konkret heisst dies:

- Labels beginnen auf Kolonne 1 und dürfen maximal 7 Stellen lang sein.
- Befehle beginnen auf Kolonne 10.
- Parameter beginnen auf Kolonne 16.
- Fortsetzungen werden mit einem Komma gekennzeichnet und müssen auf Kolonne 72 ein Fortsetzungszeichen aufweisen. Die Fortsetzung erfolgt auf der nächstfolgenden Zeile ab Kolonne 16.

Es dürfen nur die hier dokumentierten Befehle verwendet werden. Damit ist sichergestellt, dass die definierten Dialoge bei Weiterentwicklungen kompatibel sind.

```
         1     1   2         3         4         5         6         7 7
1...+....0....+6...0....+....0....+....0....+....0....+....0....+....0.2.

LABEL    BEFEHL PARAMETER1,                                            C
                PARAMETER2,                                            C
                PARAMETER3
```

Abbildung 1. Allgemeine Notations-Regeln

Beispiel:

```
         1     1   2         3         4         5         6         7 7
1...+....0....+6...0....+....0....+....0....+....0....+....0....+....0.2
_____________________________________________________________________________

BILD10   MAP
         FILL  DATA='LOGOFF',                                          C
               LIN=24,COL=05
         MAPEND
         MOVE  TO=FELD1,DATA='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXC
               XXXXXXXXX'
```

Abbildung 2. Beispiel Notations-Regeln

Bemerkung:

- **o** Alle Labels, Befehle und Parameter sind in Grosschrift einzugeben.

- **o** und Wo nicht anders vermerkt, kann der String im DATA-Parameter max. 80 Zeichen betragen kann mit FROM/TO referenziert werden, falls ein Label definiert ist.

- **o** Ein * in Kolonne 1 kennzeichnet eine Kommentar Zeile.

- **o** Sonderzeichen sind zu vermeiden; es gelten die Regeln gemäss Assembler-Konventionen.

3. ATO AUFBAU

3.1. *masc-ato* Befehlssatz

masc-ato verwendet zur Steuerung der Sessions und zur Bedienung der Anwendungsbildschirme einen übersichtlichen Funktionssatz, mit dem die verschiedensten Bildschirm- und Daten-Manipulationen gut lesbar dargestellt werden. Die einzelnen Funktionen werden unten detailliert beschrieben.

Im "native" Modus wird jeweils eine Funktion gesetzt und ein Puffer mit den Parametern vorbereitet, danach wird das Modul "ATO" aufgerufen. Für den Aufruf der Funktionen empfehlen wir die Verwendung von mnemotechnisch einprägsamen Namen, die je nach gewählter Programmiersprache in Copybooks abgelegt werden können. Für das Manual wird eine Terminologie gewählt, die speziell für die Einbettung in REXX Programme geeignet ist.

3.2. Reserved Words

Folgende Variablen sind für spezielle Zwecke reserviert:

ATO_SID	Session ID
ATO_FUNC	Funktion
ATO_RC	Return-Code
ATO_POSITION	Cursor-Position
ATO_BUFF	Parameter für eine Funktion

3.3. Tastatur und Cursorsteuerung

masc-ato kann nicht nur "normale" Zeichen an den Bildschirm senden, sondern auch die Tabulator- und die Cursor-Steuerungstasten. Dies erlaubt, auf einfache Weise änderungsfreundliche Dialoge zu schreiben.

Zum Erkennen der speziellen Eingaben dient das Zeichen "@". Dieser Escape-Character zeigt an, dass das nächste Zeichen eine besondere Bedeutung hat. Das übernächste Zeichen wird dann wieder ganz normal an den Bildschirm geschickt. Soll der Escape – Character übergeben werden, muss er wiederholt werden.

Folgende Eingaben sind möglich:

@h	**Home**	Cursor auf das erste Eingabefeld des Bildschirms stellen
@n	**Newline**	Cursor auf das erste Eingabefeld der nächsten Bildschirmzeile stellen
@r	**Cursor right**	Cursor ein Zeichen nach rechts verschieben
@l	**Cursor left**	Cursor ein Zeichen nach links verschieben
@u	**Cursor up**	Cursor eine Zeile nach oben verschieben
@d	**Cursor down**	Cursor eine Zeile nach unten verschieben
@f	**Forward Tabulator**	Cursor auf das nächste Eingabefeld stellen
@b	**Backward Tabulator**	Cursor auf das vorherige Eingabefeld stellen

Diese Spezial-Zeichen verhalten sich selbstverständlich wie die entsprechenden Tasten. Während einige Tasten nur auf Eingabefelder springen können, wird der Cursor bei anderen auch auf geschützte Bildschirmbereiche gestellt.

Selbstverständlich bewegt sich der Cursor ebenfalls entsprechend den Eingaben. Wird beispielsweise ein 8-stelliges Feld mit einem 8-stelligen Wert gefüllt, springt der Cursor automatisch auf das nächste Eingabefeld.

Die PF-, PA- und Clear-Taste werden mit dem Befehl ATO_SEND_AID abgesetzt, nachdem der Bildschirm vorher mit beliebigen Inhalten gefüllt und der Cursor auf eine bestimmte Stelle gesetzt worden ist.

3.4. Programmsteuerung

masc-ato konzentriert sich auf die Schnittstelle zu Online-Anwendungen. Die eigentliche Programmlogik wird hingegen in einer höheren Programmiersprache programmiert, die dafür besser geeignet sind und auch die nötigen Schnittstellen zu anderen System-Ressourcen wie Datenbanken, Dateien etc. aufweisen.

Nebst den Schleifen, Verzweigungen und vielfältigen Möglichkeiten zur Manipulation von Variablen kann beispielsweise aus REXX auf eine grosse Bibliothek mächtiger Funktionen zugegriffen werden, was bei der Interpretation des erhaltenen und der Vorbereitung des nächsten Bildschirms wesentliche Vorteile bringen kann.

Die oben beschriebenen Befehle werden deshalb entweder in REXX, einer trotz seiner relativen Einfachheit mächtigen, auf verschiedenen Plattformen zur Verfügung stehenden Sprache, oder in einer anderen Programmiersprache wie COBOL, Assembler oder PL/1, eingebettet.

Die Wahl der richtigen Programmiersprache für einen Dialog hängt von verschiedenen Faktoren ab. Portierbarkeit, vielfältiger Funktionen und Geschwindigkeit bei der Entwicklung sprechen bestimmt häufig für REXX, während der Zugriff auf DB/2 oder VSAM Dateien allenfalls in anderen Sprachen einfacher programmiert werden kann.

3.5. Beispiel *masc-ato*-Dialog

Die folgende Abbildung zeigt den Ablauf eines Dialoges mit ***masc-ato***-Befehlen.

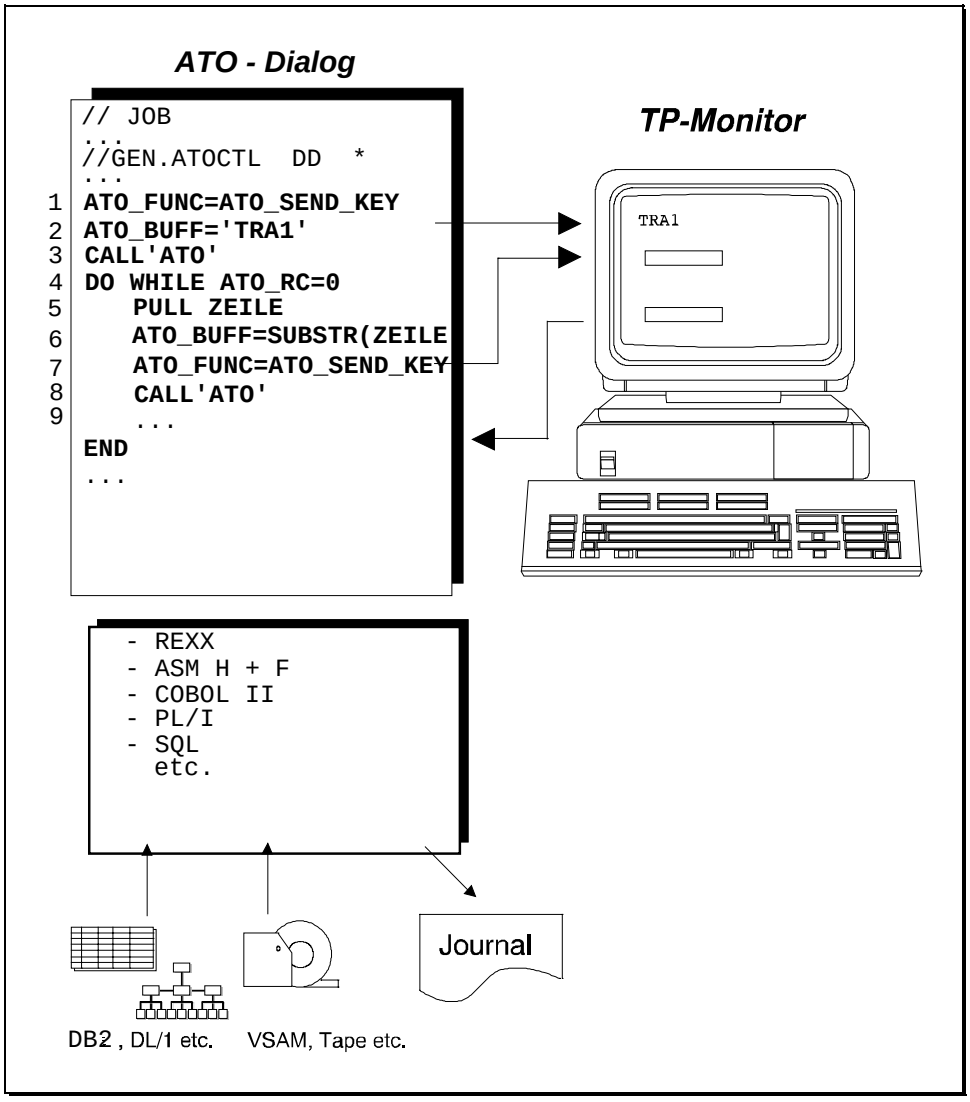


Abbildung 3. ***masc-ato*** Dialog

Erläuterungen:

1 Funktion *Tastatureingabe an den virtuellen Bildschirm senden* vorbereiten ...
2 die gewünschte Eingabe in den Puffer stellen ...
3 ... und ATO aufrufen.
4 Beginn einer Schleife mit Prüfung der Return-Codes von ATO
5 Zeile einlesen
6 Einen Teil der gelesenen Zeile in den Puffer stellen, ...
7 ... damit dieser als "Tastatureingabe" auf dem virtuellen Bildschirm kommt.
8 Aufruf von ATO
9 ... weitere ATO oder REXX Befehle und Funktionen.

Jeder ATO-Dialog beginnt mit einem PROLOG- und endet mit einem EPILOG-Befehl.

Als ATO-Dialog wird das eigentliche Session-Handling bezeichnet, während die Datenverarbeitung mit einer Programmiersprache wie REXX programmiert wird.

Jede Session beginnt mit einem SESSBEG- und endet mit einem SESSEND-Befehl. Bis zu 9 Sessions können beliebig verschachtelt und parallel durchgeführt werden, die Zuordnung der session-orientierten Befehle zu einer bestimmten Session erfolgt aufgrund des SESSID Parameters.

Sequentiell können beliebig viele Sessions verarbeitet werden.

4. ATO BEFEHLSÜBERSICHT

Dieses Kapitel zeigt die Übersicht der Befehle und Funktionen von ***masc-ato***. Soweit sinnvoll wurden der Aufruf aus REXX und der Funktionsaufruf im Kompatibilitätsmodus einander gegenübergestellt. Die detaillierte Beschreibung mit allen Parametern erfolgt später in den folgenden Kapiteln.

4.1. Dialog- und Session-Steuerung

Aufruf aus REXX	**Kompatibilitäts-Modus**	
ATO_INITIALIZE	**PROLOG**	Initialwerte eines Dialoges
ATO_TERMINATE	**EPILOG**	Ende eines ATO-Dialoges
ATO_DISCONNECT_FORCE	**ABORT**	Abbruch des Dialoges
(EXIT)	**END**	Ende
ATO_CONNECT_PS	**SESSBEG**	Beginn einer bestimmten Session Neu in ***masc-ato*** 4.1.0
ATO_DISCONNECT_PS	**SESSEND**	Beenden einer bestimmten Session Neu in ***masc-ato*** 4.1.0
ATO_QUERY_SESSION_STATUS	**n/a**	Status einer Session abfragen
ATO_SET_SESSION_PARAMETERS	**SESSSET**	Setzen der Session - Defaults Neu in ***masc-ato*** 4.1.0

Abbildung 4. Dialog- und Session - Steuerung

4.2. Bedienung Virtueller Bildschirme

4.2.1. Aufruf aus REXX

ATO_QUERY_CURSOR		Cursor-Position abfragen
ATO_SEND-KEY		Tastatureingaben an den virtuellen Bildschirm senden
ATO_SET_CURSOR		Cursor-Position setzen
ATO_SEND_AID		Enter bzw. eine PF- oder PA-Taste an den virtuellen Bildschirm senden
ATO_COPY_PS		Bildschirminhalt kopieren
ATO_COPY_PS_TO_STRING		Bildschirminhalt in einen Puffer kopieren
ATO_COPY_STRING_TO_PS		Puffer in einen virtuellen Bildschirm kopieren

Abbildung 5. Bedienung virtueller Bildschirme aus REXX

4.2.2. Kompatibilitäts-Modus

MAP	Beginn der Bildeingabe
MAPEND	Ende der Bildeingabe
FILL	Füllen eines Bildschirm-Eingabe-Feldes

MAPKEYS	Füllen eines Bildschirm-Eingabe-Feldes unter Verwendung von Cursor-Steuerungstasten Neu in ***masc-ato*** 4.1.0
MAPFLD	Füllen eines Bildschirm-Teilfeldes
MAPFILL	Füllen eines Bildschirm-Teilfeldes

Abbildung 6. Bedienung virtueller Bildschirme im Kompatibilitäts-Modus

4.3. Weitere Befehle

4.3.1. Aufruf aus REXX

ATO_PUTLOG	**PUTLOG**	Meldungen schreiben auf ATO LOG
ATO_PUTWTO	**PUTWTO**	Meldungen schreiben auf Konsole
ATO_PENDING	**PENDING**	Verarbeiten ausstehende Transaktionen
ATO_PAUSE **ATO_SLEEP**	**SLEEP**	Wartezeit
ATO_EXI	**UEXIT**	Aufruf des User Exits ATOEXI

Abbildung 7. Weitere Befehle Aufruf aus REXX

4.3.2. Kompatibilitäts-Modus

	SCAN	Suchen auf eine erhaltene Ausgabe

	SCANB	Suchen rückwärts auf eine erhaltene Ausgabe
	GETRDR	Lesen von JCL-Eingabe-Daten
ATO_PUTLOG	**PUTLOG**	Meldungen schreiben auf ATO LOG
	PUTPRT	Meldungen schreiben auf ATO PRT
ATO_PUTWTO	**PUTWTO**	Meldungen schreiben auf Konsole
	DCL	Deklarieren von Feldern
	GOTO	Verzweigen auf eine Sprung-Marke
	LOOP	Neusetzen des LOOP Counters auf einen bestimmten Wert
	MARK	Sprung-Marke
	PERFORM	Ausführen einer Prozedur
	PROC	Beginn einer Prozedur
	PROCEND	Abschluss einer Prozedur
	COPY	Kopieren von vorgegebenen Dialogsequenzen (Copybook)
	HLLCALL	Kommunizieren via HLL Interface Area mit einem aufrufenden User Programm
	LOGTIME	Definiert die Verzögerungszeit in LOGTERM
	MOVE	Übertragen
ATO_PENDING	**PENDING**	Verarbeiten ausstehende Transaktionen

ATO_PAUSE **ATO_SLEEP**	**SLEEP**	Wartezeit
ATO_EXI	**UEXIT**	Aufruf des User Exits ATOEXI
	ATOHLL	HLL Interface zum Aufrufen eines ATO Dialoges aus einem User Batch Programm mit CALL. entfällt

Abbildung 8. Weitere Befehle Kompatibilitäts-Modus

5. *masc-ato* BEFEHLE

5.1. ATO_CONNECT_PS

Syntax:

ATO_FUNC	**ATO_CONNECT_PS**
ATO_SID	*sessid*

sessid **Gültige Werte:** Maximal 1-stelliger Character

sessid zeigt, für welche Session SESSBEG gilt.

Default: Gleich wie im ATO_SET_SESSION_PARAMETERS

Beschreibung:

Der ATO_CONNECT_PS Befehl bestimmt den Beginn einer ATO Session. VORHER werden mit ATO_SET_SESSION_PARAMETERS die Parameter für die Session festgelegt.

Beispiel:

```
ATO_FUNC = ATO_CONNECT_PS
CALL 'ATO'
IF C2D(ATO_RC) > 0 THEN DO
   SAY 'SESSION COULD NOT BE ESTABLISHED, RC:' C2D(ATO_RC)
   EXIT 16
END
```

Abbildung 9. Beispiel ATO_CONNECT_PS

5.2. ATO_COPY_PS

Syntax:

ATO_FUNC	**ATO_COPY_PS**
ATO_BUFF	*area*

area **Gültige Werte:** Beliebiger Bereich

area stellt einen Bereich zur Verfügung, in den der virtuelle Bildschirm kopiert werden kann.

Beschreibung:

Diese Funktion kopiert einen virtuellen Bildschirminhalt Um Teile des virtuellen Bildschirms zu kopieren, sollte ATO_COPY_PS_TO_STRING verwendet werden.

Beispiel:

```
ATO_FUNC = ATO_COPY_PS
ATO_BUFF = COPIES(' ',4000)
CALL 'ATO'
```

Abbildung 10. Beispiel ATO_COPY_PS

5.3. ATO_COPY_PS_TO_STRING

Syntax:

ATO_FUNC	**ATO_COPY_PS_TO_STRING**
ATO_BUFF	

Beschreibung:

Mit diesem Befehl wird der virtuelle Bildschirminhalt in einen Puffer kopiert. Dort steht er für die Weiterverarbeitung zur Verfügung. Die Länge des Puffers sollte so gewählt werden, dass mindestens der ganze Bildschirm darin Platz hat. Um unvorhergesehende Resultate zu vermeiden, sollte der Puffer vor dem Aufruf initialisiert werden.

Eine Anwendung des ATO_COPY_PS_TO_STRING-Befehles ist in Abbildung 11 dargestellt. In dem Beispiel wird der Bildschirminhalt kopiert, um ihn anschliessend zeilenweise anzuzeigen. Diese Routine kann verwendet werden, um ein Hardcopy des Bildschirms auszugeben.

Beispiel:

```
ATO_FUNC = ATO_COPY_PS_TO_STRING
ATO_BUFF = COPIES(' ', 4000)
ATO_POSITION = D2C(1,4)
CALL 'ATO'
CALL RETTEST
DO WHILE POS < LENGTH(ATO_BUFF)
   SAY SUBSTR(ATO_BUFF,POS,80)
   POS = POS + 80
END
SAY '*** END OF SCREEN ***'
```

Abbildung 11. Beispiel ATO_COPY_PS_TO_STRING

5.4. ATO_COPY_STRING_TO_PS

Syntax:

ATO_POSITION	*position*
ATO_FUNC	**ATO_COPY_STRING_TO_PS**
ATO_BUFF	

Beschreibung:

Diese Funktion dient dazu, einen Puffer in einen virtuellen Bildschirm zu kopieren. Damit kann der Bildschirm mit bestimmten Inhalten gefüllt werden, um ihn anschliessend mit ATO_SEND_AID zurückzugeben.

Eine Anwendung des ATO_COPY_STRING_TO_PS-Befehles ist in Abbildung 12 dargestellt. Damit wird auf Zeile 24, Spalte 5 der String *nend* in den Bildschirm-Puffer kopiert und anschliessend abgeschickt.

Beispiel:

```
POS = 23 * 80 + 5
ATO_POSITION = D2C(POS,4)
ATO_FUNC = ATO_COPY_STRING_TO_PS
ATO_BUFF = 'nend'
CALL 'ATO'
ATO_FUNC = ATO_SEND_AID
ATO_BUFF = 'ENTER'
CALL 'ATO'
```

Abbildung 12. Beispiel ATO_COPY_STRING_TO_PS

5.5. ATO_DISCONNECT_FORCE

Syntax:

ATO_FUNC	**ATO_DISCONNECT_FORCE**
ATO_BUFF	

Beschreibung:

ATO_DISCONNECT_FORCE bricht die Verbindung zur laufenden Session ab, der Dialog bzw. das Programm läuft jedoch weiter. ATO_DISCONNECT_FORCE kann verwendet werden, um eine Weiterverarbeitung nach einem Fehler zu verhindern. Der Return-Code für den Job wird nicht durch ATO_DISCONNECT_FORCE, sondern mit dem EXIT später im Programm gesetzt.

Eine Anwendung des ATO_DISCONNECT_FORCE-Befehles ist in Abbildung 13 dargestellt.

Beispiel:

```
ATO_FUNC = ATO_DISCONNECT_FORCE
ATO_BUFF = ''
CALL 'ATO'
EXIT 16
      :
```

Abbildung 13. Beispiel ATO_DISCONNECT_FORCE

Bemerkung:

Mit ATO_DISCONNECT_FORCE erfolgt keine Abmeldung vom TP-Monitor. Dieser Befehl kommt einem Ausschalten des logischen Bildschirmes gleich. Daher sollte man diesen Befehl nach Möglichkeit vermeiden.

5.6. ATO_DISCONNECT_PS

Syntax:

ATO_SID	*sessid*
ATO_FUNC	**ATO_DISCONNECT_PS**
ATO_BUFF	

sessid **Gültige Werte:** Maximal 1-stelliger Character

Mit diesem Parameter wird angegeben, welche Session beendet werden soll.

Default: Gleich wie *sessid* aus SESSBEG.

Beschreibung:

Der ATO_DISCONNECT_PS Befehl bestimmt das Ende einer einzelnen Session. Vor dem EPILOG Befehl müssen sämtliche ATO Sessions durch ATO_DISCONNECT_PS beendet werden.

Beispiel:

```
ATO_SID = 'A'
ATO_FUNC = ATO_DISCONNECT_PS
CALL 'ATO'
```

Abbildung 14. Beispiel ATO_DISCONNECT_PS

5.7. ATO_EXI

Syntax:

ATO_FUNC	**ATO_EXI**
ATO_BUFF	

Beschreibung:

Mit der Funktion ATO_EXI kann der User-Exit ATOEXI aufgerufen werden. Der Exit bestimmt den Übergabebereich.

Wenn ATO_BUFF mit der Konstanten #VARx# gefüllt wird, enthält er anschliessend den Wert, der in der Tabelle des Programms ATOEXI abgelegt ist.

Eine Anwendung des ATO_EXI-Befehles ist in Abbildung 15 dargestellt. #VAR2# wird mit dem Wert aus der ATOEXI-Tabelle gefüllt und anschliessend als Passwort auf dem Bildschirm eingegeben.

Beispiel:

```
ATO_FUNC = ATO_EXI
ATO_BUFF = '#VAR2#'
CALL 'ATO'
#VAR2# = ATO_BUFF
ATO_FUNC = ATO_SEND_KEY
ATO_BUFF = '@Huserid@H@N'#VAR2#
CALL 'ATO'
```

Abbildung 15. Beispiel ATO_EXI

5.8. ATO_INITIALIZE

Syntax:

ATO_FUNC	**ATO_INITIALIZE**
ATO_BUFF	**TRACE=***trace*

trace **Gültige Werte:** NO, YES

Dieser Parameter dient zum Ein- bzw. Ausschalten des ATOLOGs. Wenn TRACE=YES gesetzt ist, werden alle Bildschirme angezeigt, wie sie im Dialog durchlaufen werden.

Default: YES

Beschreibung:

Der ATO_INITIALIZE Befehl bestimmt den Beginn eines ATO Dialoges und ist damit **zwingend** die erste zu verarbeitende ATO-Anweisung. Durch entsprechende Parameter wird mitgeteilt, welcher TP-Monitor als Default gilt und mit welchem logischen Terminal die Verarbeitung erfolgen soll. Weiter kann mit den jeweiligen Parametern ein detailliertes LOG oder ein zum Dialog parallel laufendes LOG-Terminal angefordert werden, welches ebenfalls zur Diagnose im Support-Center des Lizenzgebers verwendet wird.

Beispiel:

```
ATO_FUNC = ATO_INITIALIZE
ATO_BUFF = 'TRACE=YES'
CALL 'ATO'
```

Abbildung 16. Beispiel ATO_INITIALIZE

5.9. ATO_PAUSE

Syntax:

ATO_FUNC	**ATO_PAUSE**
ATO_BUFF	

Beschreibung:

ATO_PAUSE unterbricht den Dialog, bis der nächste Schirm vom Host eintrifft. Damit kann auf bestimmte Ereignisse wie beispielsweise asynchrone Meldungen, dass ein Prozess geendet hat, "gewartet" werden.

Wenn AUTOPEND im ATO_SET_SESSION_PARAMETERS nicht angegeben wurde oder auf NO steht, kann der neue Bildschirminhalt mittels einem ATO_PENDING dem Dialog zur Verfügung gestellt werden. Mit AUTOPEND=YES wird der virtuelle Bildschirm sofort mit dem neuen Inhalt gefüllt, es ist keine weitere Aktion notwendig.

Die Funktion ATO_PAUSE wird nur abgebrochen, wenn tatsächlich eine asynchrone Meldung eintrifft oder wenn ein Timeout auftritt. Falls der Dialog nur unterbrochen werden soll, um anschliessend eine Aktion zu wiederholen, sollte ATO_SLEEP verwendet werden.

SAP: Das Ende einer Langläufer-Task kann durch ATO_PAUSE abgewartet werden.

Eine Anwendung des ATO_PAUSE-Befehles ist in Abbildung 17 dargestellt. Nach der Auflösung des ATO_PAUSE wird geprüft, ob nun auf der ersten Zeile ab Spalte 20 ein bestimmter String steht.

Beispiel:

```
ATO_FUNC = ATO_PAUSE
CALL 'ATO'
ATO_FUNC = ATO_PENDING    /* nur notwendig, falls AUTOPEND=NO */
CALL 'ATO'
ATO_FUNC = ATO_COPY_PS_TO_STRING
ATO_BUFF = COPIES(' ',32)
ATO_POSITION = D2C(1,20)
CALL 'ATO'
IF ATO_BUFF = 'Titelzeile des neuen Bildschirms' THEN
  DO
    ...
  END
```

Abbildung 17. Beispiel ATO_PAUSE

5.10. ATO_PENDING

Syntax:

ATO_FUNC	**ATO_PENDING**
ATO_BUFF	

Beschreibung:

Falls in einem Dialog ausserordentliche Meldungen empfangen werden, können diese mit dem Parameter AUTOPEND=NO im ATO_SET_SESSION_PARAMETERS momentan unterdrückt werden, damit der normale Dialogablauf nicht unterbrochen wird. Mit ATO_PENDING werden diese Meldungen abgeholt, d.h., der neue Bildschirminhalt steht nun zur Verfügung und kann beispielsweise mit ATO_COPY_PS_TO_STRING in einen Arbeitsbereich kopiert werden.

Eine Anwendung des ATO_PENDING-Befehles ist in Abbildung 18 dargestellt.

Bemerkung:

SAP: Wenn eine Task zum Langläufer wird, erscheint die Meldung "Verarbeitung läuft ...". Zu einem späteren Zeitpunkt folgt der nächste Bildschirm. Dies wird durch die ATO_PAUSE erkannt, abhängig vom AUTOPEND-Parameter muss nun ein ATO_PENDING folgen, damit der neue Bildschirm dem Dialog zur Verfügung steht.

Beispiel:

```
ATO_FUNC = ATO_PENDING
CALL 'ATO'
```

Abbildung 18. Beispiel ATO_PENDING

5.11. ATO_PUTLOG

Syntax:

ATO_FUNC	**ATO_PUTLOG**
ATO_BUFF	*meldung*

Beschreibung:

ATO_PUTLOG schreibt den mitgegebenen Text auf ATOLOG.

Bemerkung:

VSE: PUTLOG schreibt auf SYSLST.

MVS: PUTLOG schreibt auf die DD-Karte ATOLOG.

Um Meldungen abzusetzen, die nur der Kontrolle eines Dialogs dienen, d.h. reine Informationshinweise, ist der SAY Befehl vorzuziehen.

Eine Anwendung des ATO_PUTLOG-Befehles ist in Abbildung 19 dargestellt.

Beispiel:

```
ATO_FUNC = ATO_PUTLOG
ATO_BUFF = 'Dieser Text erscheint im LOG'
CALL 'ATO'
```

Abbildung 19. Beispiel ATO_PUTLOG

5.12. ATO_PUTWTO

Syntax:

ATO_FUNC	**ATO_PUTWTO**
ATO_BUFF	

Beschreibung:

PUTWTO schreibt den mitgegebenen Text auf die Konsole.

Eine Anwendung des ATO_PUTWTO-Befehles ist in Abbildung 20 dargestellt.

Beispiel:

```
ATO_FUNC = ATO_PUTWTO
ATO_BUFF = 'Dieser Text erscheint auf der Konsole'
CALL 'ATO'
```

Abbildung 20. Beispiel ATO_PUTWTO

5.13. ATO_QUERY_CURSOR

Syntax:

ATO_POSITION	
ATO_FUNC	**ATO_QUERY_CURSOR**
ATO_BUFF	

Beschreibung:

ATO_QUERY_CURSOR dient der Abfrage der Cursor-Position. Diese Funktion stellt die aktuelle Cursor-Position auf ATO_POSITION, wobei der ganze Bildschirminhalt als ein String betrachtet wird. Daraus lassen sich einfach die aktuelle Zeilen- und Spalten-Position ausrechnen, dies ist in Abbildung 21 dargestellt.

Beispiel:

```
ATO_FUNC = ATO_QUERY_CURSOR
ATO_BUFF = ''
CALL 'ATO'
LINE = (C2D(ATO_POSITION) % 80) + 1
COL  = C2D(ATO_POSITION) // 80
SAY 'ATO_POSITION:' C2D(ATO_POSITION)
SAY 'LINE        :' LINE
SAY 'COL.........:' COL
```

Abbildung 21. Beispiel ATO_QUERY_CURSOR

5.14. ATO_SEND_AID

Syntax:

ATO_FUNC	**ATO_SEND_AID**
ATO_BUFF	*key*

key **Gültige Werte:** ENTER, CLEAR, PA1 - PA3, PF1 - PF24

Dieser Parameter bezeichnet die Funktionstaste bzw. die Enter-Eingabe.

Default: ENTER

Beschreibung:

ATO_SEND_AID dient dazu, eine Funktionstaste an den Bildschirm zu senden. Häufig folgt diese Funktion einem ATO_SEND_KEY.

Abbildung 22 zeigt einfache Funktionen mit ATO_SEND_AID Befehlen. So kann der Dialog übersichtlich gestaltet werden.

Beispiel:

```
CALL CLEAR
ATO_FUNC = ATO_SEND_KEY
ATO_BUFF = 'CEMT'
CALL 'ATO'
CALL ENTER

CLEAR:
ATO_FUNC = ATO_SEND_AID
ATO_BUFF = 'ENTER'
CALL 'ATO'
RETURN

ENTER:
ATO_FUNC = ATO_SEND_AID
ATO_BUFF = 'ENTER'
CALL 'ATO'
RETURN
```

Abbildung 22. Beispiel ATO_SEND_AID

5.15. ATO_SEND_KEY

Syntax:

ATO_FUNC	**ATO_SEND_KEY**
ATO_BUFF	*key*

key **Gültige Werte:** Beliebige Folge von Tastatureingaben

Beschreibung:

Mit ATO_SEND-KEY werden Tastatureingaben an den Bildschirm geschickt. Die Einbettung von speziellen Tasten mittels des Escape-Characters @ ist oben beschrieben. Falls die Eingabe mit Text beginnt, wird sie an der aktuellen Cursor-Position gemacht.

Eine Anwendung des ATO_SEND_KEY-Befehls ist in Abbildung 23 dargestellt. @H stellt zunächst den Cursor auf das erste Eingabefeld, wo der String *userid* eingegeben wird. Nach @N (newline), d.h., auf dem ersten Eingabefeld auf einer Zeile weiter unten, wird dann *password* eingegeben. Selbstverständlich könnten diese Eingaben auch mit zwei ATO_SEND_KEY Aufrufen gemacht werden, falls dies der Übersichtlichkeit dient. Bemerkung: Nach dem Aufruf des Bildschirms steht der Cursor auf dem ersten Eingabefeld, das @H könnte deshalb auch weggelassen werden.

Beispiel:

```
ATO_FUNC = ATO_SEND_KEY
ATO_BUFF = '@Huserid@Npassword'
CALL 'ATO'
ATO_FUNC = ATO_SEND_AID
ATO_BUFF = 'ENTER'
CALL 'ATO'
```

Abbildung 23. Beispiel ATO_SEND_KEY

5.16. ATO_SET_CURSOR

Syntax:

ATO_FUNC	**ATO_SET_CURSOR**
ATO_POSITION	*pos*

pos **Gültige Werte:** Abhängig vom Modell des virtuellen Bildschirms

pos bestimmt die neue Cursorposition.

Beschreibung:

Nach dem Empfang eines Bildschirms steht der Cursor auf dem ersten Eingabefeld. Falls die Tastatureingabe in ein anderes Feld gemacht werden soll, kann die Cursorposition mittels ATO_SET_CURSOR verändert werden.

Die allgemeine Formel zur Ermittlung der Cursorposition ist wie folgt:

(Zeilen-Nummer - 1) * Anzahl Zeichen pro Zeile + Spalten-Nummer

Bemerkung

Zum Positionieren auf dem Bildschirm dienen ebenfalls die "virtuellen Tastatureingaben" @f (Forward Tabulator), @n (new line) etc., die in den ATO_SEND_KEY eingebettet werden können.

Als Beispiel für den ATO_SET_CURSOR-Befehl ist in Abbildung 24 dargestellt, wie auf Zeile 24, Spalte 5 die SAP-Transaktion ABAP aufgerufen wird, wobei angenommen wird, dass dem Dialog ein Bildschirm Modell 2 mit 80 Zeichen pro Zeile zugrunde liegt.

Beispiel:

```
LIN = 24
COL = 5
POS = ((LIN - 1) * 80 + COL
ATO_FUNC = ATO_SET_CURSOR
ATO_POSITION = D2C(POS,4)
CALL 'ATO'
ATO_FUNC = ATO_SEND_KEY
ATO_BUFF = 'NABAP'
CALL 'ATO'
```

Abbildung 24. Beispiel ATO_SET_CURSOR

5.17. ATO_SET_SESSION_PARAMETERS

Syntax:

ATO_SID	*sessid*
ATO_FUNC	**ATO_SET_SESSION_PARAMETERS**
ATO_BUFF	**DIALOG=***name* ,**NETNAME=***netname* ,**APPLID=***applid* ,**APPLTRY=***appltry* ,**AUTOPEND=***autopend* ,**LOGAID=***logaid* ,**LOGTERM=***logterm* ,**LOGTIME=***logtime* ,**LOGTRY=***logtry* ,**LOOP=***loop* ,**LINEOV=***lineov* ,**MDTAUTO=***mdtauto* ,**TRACE=***trace* ,**TIMEOUT=***timeout* ,**SESSMAX=***sessmax* ,**SUPPORT=***support* ,**LOGMODE=***logmode* ,**LOGTMOD=***logtmod* **,ESC=***esc*

Abbildung 25. ATO_SET_SESSION_PARAMETERS, Syntax

sessid **Gültige Werte:** Maximal 1 Character

Dieser Parameter kennzeichnet die Session.

Default: A

name **Gültige Werte:** Maximal 8 alphanumerische Zeichen

Dieser Parameter bezeichnet den Namen des auszuführenden Dialoges.

Default: ATODLOG

netname **Gültige Werte:** Maximal 8-stelliger Terminalname gemäss Netzwerk-Konventionen

CICS: Dieser Parameter bezeichnet einen gültigen im CICS definierten oder im AUTOINSTALL zugelassenen Netzwerknamen.

Default: NETATO1

applid **Gültige Werte:** APPLID Name gemäss Netzwerk-Konventionen
APPLID für eine ATO Session.

CICS: Dieser Parameter bezeichnet den in der SIT des entsprechenden CICS eingetragenen Applikationsnamen**.** Dieser ist auch mit der Transaktion CEMT INQ TAS am unteren rechten Bildrand ersichtlich.

Default: DBDCCICS

appltry **Gültige Werte:** 1 - 6000

appltry ist die Anzahl der Versuche, die ATO für die Herstellung einer Verbindung zur Applikation in Parameter APPLID durchführt.

Die Angabe von **APPLTRY=1** kann verwendet werden, um einen Dialog nur dann auszuführen, wenn der TP-Monitor aktiv und verfügbar ist.

Default: 6000

autopend **Gültige Werte:** YES, NO

autopend bestimmt, ob eine auserordentliche Meldung, die beispielsweise durch ein EXEC CICS START TRANSID oder eine CMSG-Transaktion ausgelöst wurde, den normalen Dialogablauf unterbricht, oder ob sie für den Moment zurückgestellt werden soll.

Die Angabe von NO erlaubt, die Meldungen an vordefinierten Stellen mit ATO_PENDING dem Dialog verfügbar zu machen, um sie anschliessend zu verarbeiten.

Default: NO

logaid **Gültige Werte:** YES oder NO.

Wenn dieser Parameter auf YES steht, wird beim Verfolgen der Session auf dem LOGTERM unten rechts der Key angezeigt, der gedrückt wurde. Damit wird das Finden von Fehlern mit Hilfe von LOGTERM vereinfacht.

Steht LOGTERM auf NO, hat der Wert von LOGAID keinen Einfluss.

Default: NO

logterm **Gültige Werte:** Maximal 8 alphanumerische Zeichen. Gültiger Netzname gemäss Netzwerk-Konventionen, NO oder LOGON.

Dieser Parameter definiert den Netzwerknamen eines Bildschirms für die Überwachung und Schritt-für-Schritt-Anzeige dieses Dialoges. Ist diese Anzeige nicht erwünscht, so ist LOGTERM=NO zu spezifizieren.

Bei Angabe von LOGTERM=LOGON kann man sich dynamisch in den LOGTERM einschalten durch Eingabe von LOGON APPLID(*netname*). ATO wartet dabei mit der Verarbeitung , bis ein LOGON APPLID(*netname*) erfolgt.

Bemerkung:

LOGTERM ist auch für Session-Manager Produkte unterstützt.

Die Verwendung von LOGON APPLID(...) entspricht dem Default USSCMD-FORMAT=PL1 in der USSTAB Definition. Bei Angabe von FORMAT=BAL ist LOGON APPLID=... zu verwenden. Die Eingabe erfolgt bei USSTAB Message 10.

Default: NO

logtime **Gültige Werte:** 0 - 60

logtime definiert eine Zeitverzögerung in Sekunden mit LOGTERM. Während dieser Zeit wartet ATO mit der Ausgabe des nächsten Bildes.
Mit dem Befehl LOGTIME kann der Wert im Dialog neu gesetzt und übersteuert werden (Siehe Befehl LOGTIME auf Seite 63).

Default: 2

logtry **Gültige Werte:** 0 - 6000

logtry definiert die Anzahl der Versuche, um eine Verbindung zum LOGTERM Bildschirm herzustellen. Bei Angabe von **LOGTRY=1** wird der LOGTERM Bildschirm nur verwendet, wenn dieser auch verfügbar ist bzw. dieser im VTAM definiert ist. Ansonsten wird die Verarbeitung ohne LOGTERM fortgesetzt.

Default: 6000

loop **Gültige Werte:** 1 - 100000

loop definiert den Maximalwert, den der Loop-Counter erreichen kann. Beim Erreichen dieses Werts wird der Dialog abgebrochen.

Default: 9000

lineov **Gültige Werte:** 1 - 99999

Mit *lineov* wird die Anzahl Zeilen pro Seite bestimmt.

Default: 55

mdtauto **Gültige Werte:** YES, NO

Bei Angabe von MDTAUTO=YES werden für MDT-Bildschirmfelder automatisch FILL-Befehle generiert (z.B. CICS BMS-FSET Felder). Damit können auch Felder übertragen werden, die für den Benutzer auf dem Bildschirm unsichtbar (Dark) bleiben und nur der Dialogsteuerung dienen.

Bei Definition von MDTAUTO=NO werden nur Felder übertragen, die mit MAPFILL eingefügt oder verändert wurden.

Default: YES

timeout **Gültige Werte:** 0 - 60000

Dieser Parameter definiert die maximale Zeit in Sekunden, während der ATO auf eine Antwort wartet. Dieser Parameter setzt den Default für MAP- und PENDING-TIMEOUT. Der Default-Wert genügt für die meisten Anwendungen.

Mit der Angabe von TIMEOUT=1440 wird die Zeitüberschreitungs-Kontrolle ausgeschaltet. TIMEOUT=1440 sollte nur für langlaufende Dialoge verwendet werden.

Default: 120

trace **Gültige Werte:** NO, YES

Dieser Parameter dient zum Ein- bzw. Ausschalten des ATOLOGs. Wenn TRACE=YES gesetzt ist, werden alle Bildschirme angezeigt, wie sie im Dialog durchlaufen werden.

Default: YES

sessmax **Gültige Werte:** 1 - 9

Dieser Parameter gibt die Anzahl Sessions an, die in einem Dialog gleichzeitig aktiv sein können.

Default: 1

support **Gültige Werte:** NO, YES

Dieser Parameter dient der Diagnose-Hilfe. Bei Verwendung von SUPPORT=YES werden detaillierte Dialogauswertungen auf den ATOLOG geschrieben. Dabei werden auch die Werte aus dem User Exit ATOEXI ausgegeben.

Default: NO

VSE: Bei SUPPORT=YES muss der JCL-Parameter //OPTION LOG,PARTDUMP definiert sein.

MVS: Bei SUPPORT=YES muss der JCL-Parameter MSGLEVEL=(1,1), sowie //SYSUDUMP definiert sein.

logmode **Gültige Werte:** LOGMODE Name gemäss VTAM Definition

Mit diesem Parameter wird spezifiziert, welcher LOGMODE für den virtuellen Bildschrim gelten soll.

Es können alle Bildschirm-Modelle (Modell 2, 3, 4 oder 5) verwendet werden.

Default: D4A32782

logtmod **Gültige Werte:** LOGMODE Name gemäss VTAM Definition

Mit diesem Parameter wird spezifiziert, welcher LOGMODE für den LOGTERM-Bildschrim gelten soll. In der Regel sollte derselbe LOGMODE wie beim *logmode* Parameter angegeben werden.

Es können alle Bildschirm-Modelle (Modell 2, 3, 4 oder 5) verwendet werden.

Default: D4A32782

esc **Gültige Werte:** Maximal 1 Character

Dieser Parameter kennzeichnet den Escape - Character, der zur Kennzeichnung der Tastatur und Cursorsteuerung dient.

Default: @

Beschreibung:

Der ATO_SET_SESSION_PARAMETERS Befehl bestimmt den Beginn einer ATO Session. Durch entsprechende Parameter wird mitgeteilt, welcher TP-Monitor als Default gilt und mit welchem logischen Terminal die Verarbeitung erfolgen soll. Weiter kann mit den jeweiligen Parametern ein detailliertes LOG oder ein zum Dialog parallel laufendes LOG-Terminal angefordert werden, welches ebenfalls zur Diagnose im Support-Center des Lizenzgebers verwendet wird.

Das Zusammenspiel der PROLOG-Definitionen mit denjenigen der Applikationen und von VTAM ist im "*Installation Manual*" beschrieben.

Bemerkungen:

Der TIMEOUT-Parameter bestimmt zusätzlich die Einstellung auf die zu erwartende Antwortzeit.

Wird bei einer MAP-Eingabe eine längere Antwortzeit erwartet als diejenige des TIMEOUT-Parameters, muss der MAP resp. PENDING TIMEOUT-Parameter entsprechend gesetzt werden. Damit wird der Zeitwert für diese Eingabe lokal definiert.

Der TIMEOUT-Wert sollte nicht unnötig erhöht werden, da ATO sonst keine internen logische Fehler erkennen kann. Für einzelne Eingaben mit erwartungsgemäss längerer Antwortzeit ist der MAP TIMEOUT-Parameter zu verwenden.

Der APPLTRY-Parameter wird verwendet, um einen Dialog nur dann durchzuführen, wenn der TP-Monitor verfügbar ist. Zu diesem Zweck definieren Sie APPLTRY=1.

Beispiel:

```
ATO_SID = 'A'
ATO_FUNC = ATO_SET_SESSION_PARAMETERS
ATO_BUFF =          'DIALOG=SAMPLE1,'
ATO_BUFF = ATO_BUFF 'NETNAME=NETATO1,'
ATO_BUFF = ATO_BUFF 'APPLID=DBDCCICS,'
ATO_BUFF = ATO_BUFF 'LOGTERM=LOGON'
CALL 'ATO'
```

Abbildung 26. Beispiel ATO_SET_SESSION_PARAMETERS

5.18. ATO_SLEEP

Syntax:

ATO_FUNC	**ATO_SLEEP**
ATO_POSITION	*sec*
ATO_BUFF	

sec **Gültige Werte:** 1 - 3600.

Zeitwert in Sekunden für den ATO_SLEEP Befehl.

Beschreibung:

ATO_SLEEP dient zur Unterbrechung des laufenden Dialoges. Durch einen ATO_SLEEP-Befehl wird kein neuer Bildschirm eingelesen.

Eine Anwendung des ATO_SLEEP-Befehles ist in Abbildung 27 dargestellt.

Beispiel:

```
ATO_FUNC = ATO_SLEEP
ATO_POSITION = D2C(10,4)     /* 10 Sekunden warten    */
CALL 'ATO'
```

Abbildung 27. Beispiel ATO_SLEEP

5.19. ATO_TERMINATE

Syntax:

ATO_FUNC	**ATO_TERMINATE**
ATO_BUFF	

Beschreibung:

ATO_TERMINATE dient zum Abschluss eines ATO Dialoges und darf nur **einmal** am Ende eines Dialoges verwendet werden.

Vor dem ATO_TERMINATE müssen alle Sessions, die durch SESSBEG eröffnet wurden, mit dem Befehl SESSEND beendet werden, andernfalls wird ein Return-Code von 8 gesetzt.

Ein Anwendungsbeispiel für diesen Befehl ersehen Sie aus Abbildung 28.

Beispiel:

```
ATO_FUNC = ATO_TERMINATE
ATO_BUFF = ''
CALL 'ATO'
EXIT 16
       :
```

Abbildung 28. Beispiel ATO_TERMINATE

6. KOMPATIBILITÄTS-MODUS

6.1. Interne Work-Bereiche

Die Work-Bereiche WORK1 - 9 sind Arbeitsbereiche, die durch Befehle mit Feldparametern referenziert werden. Jeder dieser Bereiche ist 80 Bytes lang und mit SPACES initialisiert. Sie können in den TO= und FROM= Parameter der entsprechenden Befehle verwendet werden.

Folgende Befehle können die Work-Bereiche WORK1 - 9 ansprechen:

- **HLLCALL** (Default: TO=WORK1)
- **PUTPRT** (Default: FROM=WORK1)
- **PUTLOG** (Default: FROM=WORK1)
- **PUTWTO** (Default: FROM=WORK1)
- **SCAN** (Default: TO=WORK1)
- **SCANB** (Default: TO=WORK1)
- **GETRDR** (Default: TO=WORK1)
- **MOVE** (Default: FROM=WORK1)
- **UEXIT** (Default: TO=WORK1)

6.2. Anwendungsbeispiele der WORK-Bereiche

```
          :
          MOVE FROM=SPACES,TO=WORK1
          MOVE DATA='AAA',TO=WORK1
          :
```

Abbildung 29. Beispiel 1 WORK-Bereiche

```
          :
          MOVE   FROM=WORK1,TO=S1
S1        SCAN   DATA='BBB',TO=WORK2
          PUTPRT FROM=WORK1
          GETRDR EOF=EOF1,TO=WORK2
          MOVE   FROM=WORK2,COL=2,TO=F1
          MAP
F1        FILL   DATA='XXXX'
          MAPEND
          SCAN   DATA='A',FROM=WORK1,TO=WORK1,FOUND=M1
          :
M1        MARK
          :
```

Abbildung 30. Beispiel 2 WORK-Bereiche

6.3. ABORT

Syntax:

	Befehl	*Parameter*
	ABORT	[**RC=***retcode*]

retcode **Gültige Werte:** 04, 08, 16

Returncode, der beim Abbruch eines Dialoges gesetzt wird. Auf diesen Returncode kann in der JCL referenziert werden und somit eine allfällige Abhängigkeit der Nachfolgeverarbeitung gesteuert werden.

Default: 08

Beschreibung:

ABORT bricht den laufenden Dialog ab. Alle diesem Befehl nachfolgenden Verarbeitungen werden nicht mehr ausgeführt. ABORT kann verwendet werden, um eine Weiterverarbeitung nach einem Fehler zu verhindern. Eine Anwendung des ABORT-Befehles ist in Abbildung 31 dargestellt.

Beispiel:

```
FATAL  MARK
       PUTPRT DATA='FATAL-FEHLER'
       ABORT RC=16
       :
```

Abbildung 31. Beispiel ABORT

Bemerkung:

Mit ABORT erfolgt keine Abmeldung vom TP-Monitor. Dieser Befehl kommt einem Ausschalten des logischen Bildschirmes gleich. Als Alternative kann die in Abbildung 32 dargestellte Dialog-Sequenz verwendet werden.

```
FATAL  MARK
       PUTPRT DATA='FATAL-FEHLER'
       MAP  KEY=CLEAR
       MAPEND
       MAP LASTMAP=YES,RC=16,DATA='CESF LOGOFF'
       MAPEND
       :
```

Abbildung 32. ABORT Alternative

Konventionen für Return-Codes:

00 - Verarbeitung erfolgreich
04 - Warnung
08 - Fehler
16 - Schwerwiegender Fehler

Funktion bei Aufruf aus einer Programmiersprache:

ATO_DISCONNECT_FORCE

6.4. COPY

Syntax:

	Befehl	*Parameter*
	COPY	*name*

VSE: *name* **Gültige Werte:** Gültiger Bookname gemäss VSE Konventionen.

Das Copybook muss in einer der "// LIBDEF SEARCH"-Libraries vorhanden sein.

MVS: *name* **Gültige Werte:** Gültiger Membername gemäss MVS Konventionen.

Das Member muss in einem Dataset der "//SYSLIB"-Konkatenierung vorhanden sein.

Default: Keiner

Beschreibung:

COPY übernimmt vordefinierte Dialogsequenzen in den Ablauf. Dies entspricht in der Funktion dem "COPY", wie er in Programmiersprachen üblich ist.

Beachten Sie die Möglichkeit von FILL DATA='#VARn#' für Logon Sequenzen auf Seite 74.

Ein Anwendungsbeispiel ist in Abbildung 33 dargestellt.

Beispiel:

```
PROLOG ...
:
COPY ATOSAP
EPILOG
END
```

Abbildung 33. Beispiel COPY

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.5. DCL

Syntax:

	Befehl	*Parameter*
field	**DCL**	**DATA=**'*data*'

field **Gültige Werte:** Maximal 7-stelliger alphanumerischer Feldname

field bestimmt den Namen eines Feldes, das zur weiteren Verarbeitung definiert wird. Dieser Name ist eine **zwingende** Angabe.

Default: Keiner

data **Gültige Werte:** Maximal 80 alphanumerische Zeichen

data kann einen konstanten Wert beinhalten oder beliebige alphanumerische Zeichen. Die Anzahl der Zeichen bestimmt die Länge des Feldes. Der DATA Parameter ist eine **zwingende** Angabe.

Default: Keiner

Beschreibung:

Mit dem Befehl DCL (Declare) können Felder definiert werden, die in der weiteren Verarbeitung mit FROM/TO verwendet werden. Der Befehl kann an beliebiger Stelle im Dialog stehen. Zwecks besserer Übersichtlichkeit des Dialoges wird jedoch empfohlen, die Deklarationen in einem eigenen Abschnitt am Anfang des Dialoges zu definieren.

Ein Anwendungsbeispiel ist in Abbildung 34 dargestellt.

Beispiel:

```
         PROLOG ...
*********************************************
* DECLARATION                               *
*********************************************
MSG01    DCL  DATA='ERROR IN LOGON PROC'
CUSTNR   DCL  DATA='123.456.789'
RETC     DCL  DATA='FALSE'
         :
*********************************************
* HAUPTVERARBEITUNG                         *
*********************************************
         PERFORM PROC=LOGON
         SCAN DATA='TRUE',FROM=RETC,FOUND=MAP20
         PUTPRT FROM=MSG01
         PERFORM PROC=ERR08
MAP20    MARK
         MOVE FROM=CUSTNR,TO=FLDCUST
         MAP
FLDCUST  FILL LIN=05,COL=20,DATA='999.999.999'
         MAPEND
         :
         MAP  KEY=CLEAR
         MAPEND
         MAP  LASTMAP=YES,DATA='CESF LOGOFF'
         MAPEND
*********************************************
* PROZEDUREN                                *
*********************************************
LOGON    PROC
         :
         MOVE DATA='TRUE',TO=RETC
         PROCEND
*
ERR08    PROC
         :
         PROCEND
         EPILOG
         END
```

Abbildung 34. Beispiel DCL

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.6. EPILOG

Syntax:

	Befehl	*Parameter*
	EPILOG	

Beschreibung:

EPILOG dient zum Abschluss eines ATO Dialoges und darf nur **einmal** am Ende eines Dialoges verwendet werden.

Vor dem EPILOG müssen alle Sessions, die durch SESSBEG eröffnet wurden, mit dem Befehl SESSEND beendet werden, andernfalls wird ein Return-Code von 8 gesetzt.

Ein Anwendungsbeispiel für diesen Befehl ersehen Sie aus Abbildung 35.

Beispiel:

```
PROLOG ...
:
EPILOG
Program Code
```

Abbildung 35. Beispiel EPILOG

Funktion bei Aufruf aus einer Programmiersprache:

ATO_TERMINATE

6.7. FILL

Syntax:

	Befehl	*Parameter*
[*label*]	**FILL**	**DATA=**'*data*' [,**LIN=***line* ,**COL=***column*]

label **Gültige Werte:** maximal 7 Zeichen

Wird dem FILL-Befehl ein Label zugeordnet, kann auf dieses Feld mit TO referenziert werden.

data **Gültige Werte:** maximal 80 Zeichen

Dieser Parameter enthält den Text, der an der definierten Position übergeben wird.

Mittels DATA='#VARn#' können Variablen aus dem User Exit ATOEXI übergeben werden. Vergleichen Sie dazu auch Kapitel "User Exit ATOEXI" auf Seite 129.

CICS: Die Verwendung von TCT-UCTRAN=YES Parameter bewirkt die Umsetzung in Grossbuchstaben.

TSO: Innerhalb von TSO sind verschiedene Umsetzungen aktiv (z.B. Codepages im ISPF etc.).

line **Gültige Werte:** 01 - 24

Dieser Parameter definiert die Bildschirmzeile, in welcher der Wert übergeben wird.

Default: 01

column **Gültige Werte:** 01 - 80

Dieser Parameter definiert die Bildschirmkolonne, in welcher der Wert übergeben wird.

Default: 01

Beschreibung:

FILL dient zur Übergabe von Bildschirmwerten an ein Eingabefeld. Die LIN=- und COL=-Parameter bestimmen die Eingabeposition. Die Eingabelänge ergibt sich aus der Anzahl Zeichen zwischen den Hochkommas. Erstreckt sich der DATA-Parameter über 2 oder mehr Zeilen, muss auf Kolonne 72 ein Fortsetzungszeichen definiert werden. Die nächste Zeile muss in diesem Fall ab Kolonne 16 beginnen. Die Notation erfolgt gemäss Assembler-Konventionen. Vergleichen Sie dazu Beispiel 2 in Abbildung 14.

Beachten Sie, dass ein FILL-Befehl nur zwischen MAP- und MAPEND-Befehlen vorkommen darf. Vergleichen Sie dazu die Beispiele in Abbildungen 36 und 37.

Anstelle des FILL-Befehls können Sie neu auch den Befehl MAPFILL verwenden. Dies erlaubt Bildschirmeingaben unter Verwendung von Tasten wie Tabulator, Home, Newline etc. Details dazu entnehmen Sie bitte dem entsprechenden Kapitel.

Bemerkung:

CICS: Die Eingabelänge wird von bestimmten Transaktionen wie z.B. CESN geprüft. Bei Eingaben von Transaktionscodes auf einem leeren Bildschirm wird mit Vorteil der Befehl MAP DATA='tran' verwendet. Dies entspricht einer Eingabe im Line-Modus.

Wird der Bildschirm mit MAP KEY=CLEAR gelöscht, so ist anschliessend MAP DATA='*data*' zu verwenden.

Bei Verwendung von neueren Bildschirmen mit LIN/COL-Anzeige können diese Zahlen direkt in die LIN- und COL-Parameter eingesetzt werden.

Beispiele:

```
BILDN   MAP
        FILL COL=5,LIN=10,DATA='EINGABE'
        MAPEND
        :
```

Abbildung 36. Beispiel 1 FILL

```
1...+....1....+....2....+....3....+....4          7
         0         0         0         0          2
_____________________________________________________________________________
         GETRDR EOF=EOF1,TO=ZEILE1
BILDN    MAP
ZEILE1   FILL  DATA='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXC
               XXXXXXXXXXXXXXXXXXXXXXXXXX'
         MAPEND
         :
```

Abbildung 37. Beispiel 2 FILL

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.8. GETRDR

Syntax:

	Befehl	*Parameter*
	GETRDR	**EOF=***mark* [,**TO=***field*]

mark **Gültige Werte:** Maximal 7 Stellen.

Der Parameter definiert eine Sprungmarke, auf die im Falle einer EOF (Ende des Files) -Bedingung verzweigt wird. *mark* muss mit dem Befehl MARK definiert sein um eine EOF-Bedingung verarbeiten zu können.

Dieser Parameter ist zwingend.

field **Gültige Werte:** Maximal 7-stelliges, definiertes Feld-Label oder
WORK1 - 9.

Dieses Feld dient zum Empfangen von Werten mit dem Befehl GETRDR. Es können maximal 80 Zeichen pro GETRDR eingelesen werden.

VSE: Die Werte werden über SYSRDR in der JCL eingelesen.

MVS: Die Werte werden über das ATORDR DD-Statement eingelesen.

Default: WORK1

Beschreibung:

GETRDR dient zum Einlesen von Werten. Wird der Default für *field* nicht übernommen, so ist ein TO-Feld zu referenzieren.

MVS: Mit der JCL-Definition //ATORDR DD DSN=*datasetname(member)* können auch PDS-Member eingelesen werden.

ATORDR hat eine DCB-Definition von DCB=(LRECL=80,BLKSIZE=80,RECFM=FB).

Beispiele:

```
        GETRDR TO=VAR1,EOF=EOF1
BILDN   MAP
VAR1    FILL COL=02,LIN=02,DATA='VARIABLE1'
        MAPEND
        :
EOF1    MARK
        PUTPRT DATA='DATEN-ENDE'
        GOTO MARK=ENDE
        :
```

GETRDR JCL-Daten:

```
MANDANT1
/*
```

Abbildung 38. Beispiel 1 GETRDR

```
        :
B10     MAP DATA='FIBU'
        MAPEND
        :
L20     MARK
        GETRDR TO=WORK2,EOF=END1
        MOVE FROM=WORK2,COL=1,TO=TEXT
        MOVE FROM=WORK2,COL=13,TO=MENGE
        MOVE FROM=WORK2,COL=21,TO=ART
B20     MAP
TEXT    FILL LIN=10,COL=14,DATA='XXXXXXXXXXX'
MENGE   FILL LIN=11,COL=14,DATA='XXXXX.XX'
ART     FILL LIN=12,COL=14,DATA='XX'
        MAPEND
        GOTO MARK=L20
END1    MARK
        :
```

GETRDR JCL-Daten:

```
0...+....1....+....2....
1        0         0
ARTIKEL1    00010.0502
ARTIKEL2    07012.7003
/*
```

Abbildung 39. Beispiel 2 GETRDR

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.9. GOTO

Syntax:

	Befehl	*Parameter*
	GOTO	**MARK=***mark*

mark **Gültige Werte:** Maximal 7 Stellen

Dieser Parameter referenziert eine durch den Befehl MARK definierte Sprungmarke oder ein *label* einer MAP Definition.

Beschreibung:

Dieser Befehl dient zur Verzweigung an ein bestimmtes *label*-MARK oder *label*-MAP.

Beispiel:

```
        :
        GOTO MARK=ENDE
        :
ENDE    MARK
        :
        GOTO MARK=BILD10
        :
BILD10  MAP
        :
        MAPEND
```

Abbildung 40. Beispiel GOTO

Bemerkung:

Setzen Sie diesen Befehl sparsam ein, um die Lesbarkeit des Dialoges für die Maintenance oder bei Fehlerdiagnosen übersichtlich zu gestalten. Versuchen Sie den Dialog möglichst sequentiell (Top-Down) mit wenigen Verzweigungen zu kodieren. Als Alternative können Sie den Befehl PERFORM einsetzen, um eine Befehlssequenz durchzuführen. Der PERFORM-Befehl ist ab Seite 83 beschrieben.

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.10. HLLCALL

Syntax:

	Befehl	*Parameter*
[*label*]	**HLLCALL**	**DATA=**'*data*' [,**TO=***field*]

label **Gültige Werte:** Maximal 7 Zeichen.

Bei Angabe eines Labels kann der Inhalt des DATA-Parameters z.B. mit einem MOVE Befehl entsprechend gefüllt werden.

Default: Keiner

data **Gültige Werte:** Maximal 80 alphanumerische Zeichen.

Dieses Feld definiert den Bereich, in den die Daten *data* gestellt werden. Es wird dem aufrufenden Programm unter dem Namen HLLDATA für die Programm-Steuerung zur Verfügung gestellt.

Default: Keiner

field **Gültige Werte:** Maximal 7-stelliges, definiertes Feld-Label oder
WORK1 - 9.

field ist ein Wert, der dem aufrufenden Batch Programm mittels dem HLLTO Bereich zur Verfügung gestellt wird.

Default: WORK1

Beschreibung:

HLLCALL dient zum Kommunizieren mit dem Batch Programm, das den Dialog mit CALL ATOHLL aufgerufen hat. Dabei können Daten über die Area des HLL Interface zur weiteren Verarbeitung zur Verfügung gestellt werden (siehe HLL Interface Area). Ein erneuter Aufruf des ATO Dialoges mit CALL ATOHLL setzt die Verarbeitung nach dem HLLCALL-Befehl fort.

Bemerkung:

HLLCALL kann nur in Dialogen verwendet werden, die mittels CALL ATOHLL ... von einem Batch Programm aufgerufen werden. Durch Angabe von PROLOG SUPPORT=YES können die übergebenen Data Areas und Parameter detailliert protokolliert und überprüft werden. HLLCALL wird implizit aufgerufen nach einem MAP LASTMAP=YES und nach dem Befehl ABORT. Nach diesen beiden Befehlen ist der ATO Dialog beendet und ein erneuter Aufruf des Dialoges mit dem HLL Interface führt zu unerwünschten Resultaten.

Beispiel 1:

```
        :
        HLLCALL TO=WORK2,DATA='CALENDAR'
        PUTPRT  FROM=WORK2
        MOVE    TO=CALENDR,FROM=WORK2
        MAP
CALENDR FILL    LIN=07,COL=14,DATA='XXX'
        MAPEND
        :
```

Abbildung 41. Beispiel 1 HLLCALL

Beispiel 2:

```
        :
L1      MARK
        HLLCALL DATA='NEXT',TO=WORK2
        PUTPRT FROM=WORK2
        SCAN   FROM=WORK2,DATA='/*',FOUND=END1
        MAP    DATA='FIBU'
        MAPEND
        MOVE   TO=KREDI,FROM=WORK2,COL=01
        MAP
KREDI   FILL   LIN=07,COL=14,DATA='00000000'
        MAPEND
        SCAN   DATA='WRONG',FOUND=L2
        GOTO   MARK=L3
L2      MARK
        MOVE   TO=WORK2,FROM=SPACES
        HLLCALL DATA='NOTFOUND',TO=WORK2
        GOTO   MARK=L1
L3      MARK
        MOVE   TO=WORK2,FROM=SPACES
        MOVE   TO=WORK3,FROM=SCREEN,LIN=08,COL=02
        HLLCALL DATA='FOUND',TO=WORK2
        GOTO   MARK=L1
END1    MARK
        MAP    LASTMAP=YES,DATA='CESF LOGOFF',RC=0
        MAPEND
```

Abbildung 42. Beispiel 2 HLLCALL

Beispiel 3:

```
        :
L1      MARK
        HLLCALL DATA='DATAENTRY',TO=WORK2
        SCAN   FROM=WORK2,DATA='/*',FOUND=L9
        MOVE   TO=TRANS,FROM=WORK2
TRANS   MAP    DATA='XXXX'
        MAPEND
        MOVE   TO=NUMBER,FROM=WORK2,COL=05
        MAP
NUMBER  FILL   LIN=02,COL=02,DATA='XXXXXX'
        MAPEND
        SCAN   DATA='WRONG',FOUND=L3
        MOVE   TO=TEXT,FROM=WORK2,COL=11
        MOVE   TO=AMOUNT,FROM=WORK2,COL=21
        MAP
TEXT    FILL   LIN=10,COL=05,DATA='XXXXXXXXXX'
        FILL   LIN=11,COL=08,DATA='BY ATOHLL'
AMOUNT  FILL   LIN=12,COL=05,DATA='0000000000'
        MAPEND
        SCAN   DATA='MUTIERT',NFOUND=L2
        MOVE   TO=WORK2,DATA='OK'
        GOTO   MARK=L1
L2      MARK
        MOVE   TO=WORK2,DATA='ERROR-DATA'
        GOTO   MARK=L1
L3      MARK
        MOVE   TO=WORK2,DATA='ERROR-KEY'
        GOTO   MARK=L1
L9      MARK
        MAP    LASTMAP=YES,DATA='CESF LOGOFF'
        MAPEND
```

Abbildung 43. Beispiel 3 HLLCALL

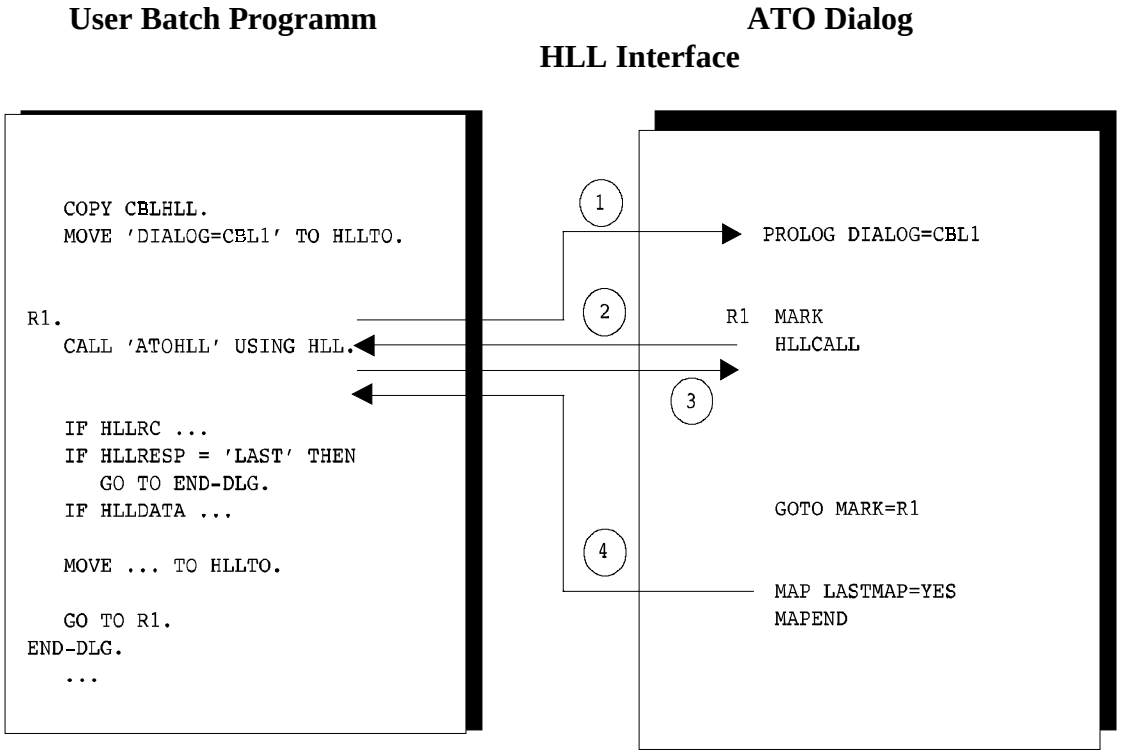
6.11. ATOHLL Interface

Beschreibung:

Das ATOHLL Interface dient dazu, einen ATO Dialog aus einem Batch Programm aufzurufen und mittels HLLCALL zu kommunizieren.

Die folgende Abbildung zeigt das Prinzip des ATOHLL Interface und den Ablauf und die Kommunikation mit dem ATO Dialog.

Die in dieser Darstellung verwendeten IF-Sätze können auch mit entsprechenden SELECT Statements dargestellt werden.



1) Mit dem ersten Aufruf von ATOHLL aus dem Batch-Programm wird der ATO Dialog initialisiert (INIT-Phase).

2) Der ATO-Dialog gibt die Kontrolle mit HLLCALL an das Batch-Programm zurück, wobei Informationen mit HLLCALL DATA= übergeben werden.

3) Der ATO-Dialog wird erneut aufgerufen mit der Übergabe von Daten im Feld HLLTO (NEXT-Phase).

Die Sequenzen 2) und 3) können so oft wiederholt werden, wie dies nötig ist.

4) LASTMAP=YES signalisiert dem Batch-Programm das Ende des ATO-Dialoges (LAST-Phase).

6.12. HLL Interface Area

Um aus einem Batch Programm mit einem ATO Dialog via HLL Schnittstelle kommunizieren zu können, wird ein Kommunikationsbereich, die sogenannte HLL Interface Area, zur Verfügung gestellt. Diese Area hat einen bestimmten Aufbau, der vom Batch Programm entsprechend gefüllt werden muss oder nach Rücksprung aus dem ATO Dialog in das Batch Programm mit entsprechenden Daten gefüllt ist. Die folgenden Felder stehen zur Verfügung:

Feldname	**Zweck**
HLLLENG	Länge der HLL Interface Area. Diese Länge ist fix 256 Bytes lang.
HLLREQU	Request Typ. Wird von ATO automatisch nachgeführt
HLLRESP	Response Typ. Wird bei LASTMAP=YES mit dem Wert LAST gefüllt.
HLLRC	Return Code Feld. Dieses Feld wird von ATO mit einem entsprechenden Return Code gefüllt bei der Rückkehr in das aufrufende Batch Programm
HLLDATA	80 Bytes langer Steuerungs-Bereich für den ATO Dialog. HLLDATA darf vom Batch-Programm nicht verändert werden.
HLLTO	80 Bytes langer Übertragungs-Bereich für das User Batch Programm

Die folgenden Abbildungen zeigen die Kodierung der HLL Interface Area für die Programmiersprachen Assembler, COBOL und PL/I, wobei dieser Bereich für andere Programmiersprachen entsprechend übernommen werden kann.

Assembler Struktur für HLL Interface Area

```
************************
*
* HLL FOR ASSEMBLER
*
************************
HLL      DS   0CL256
HLLLENG  DC   F'256'              +0
HLLID    DC   CL8'ATOHLL'         +4
HLLREQ   DC   CL4'INIT'           +12
HLLRESP  DC   CL4' '              +16
HLLRC    DC   F'8'                +20
         DC   CL8' '              +24
HLLTO    DC   CL80' '             +32
HLLDATA  DC   CL80' '             +112
         DC   CL64' '             +192
```

Abbildung 44. HLL Interface Area (Assembler Struktur)

COBOL Struktur für HLL Interface Area

```
*********************************
*
*   HLL FOR COBOL
*
*********************************
01  HLL.
    05 HLLLENG PIC S9(5) COMP VALUE +256.
    05 HLLID   PIC X(8)       VALUE 'ATOHLL'.
    05 HLLREQU PIC X(4)       VALUE SPACES.
    05 HLLRESP PIC X(4)       VALUE SPACES.
    05 HLLRC   PIC S9(5) COMP VALUE +8.
    05 FILLER  PIC X(8)       VALUE SPACES.
    05 HLLTO   PIC X(80)      VALUE SPACES.
    05 HLLDATA PIC X(80)      VALUE SPACES.
    05 FILLER  PIC X(64)      VALUE SPACES.
```

Abbildung 45. HLL Interface Area (COBOL Struktur)

PL/I Struktur für HLL Interface Area

```
/*****************************************/
   /*                                     */
   /*  HLL FOR PL/1                       */
   /*                                     */
   /*****************************************/
   DCL ATOHLL ENTRY OPTIONS(ASSEMBLER);
   DCL 1 HLL,
       DCL 2 HLLLENG FIXED BIN(31) INIT(256),
       DCL 2 HLLID   CHAR(8)       INIT('ATOHLL'),
       DCL 2 HLLREQ  CHAR(4)       INIT('INIT'),
       DCL 2 HLLRESP CHAR(4)       INIT(' '),
       DCL 2 HLLRC   FIXED BIN(31) INIT(8),
       DCL 2 HLLFIL1 CHAR(8)       INIT(' '),
       DCL 2 HLLTO   CHAR(80)      INIT(' '),
       DCL 2 HLLDATA CHAR(80)      INIT(' '),
       DCL 2 HLLFIL2 CHAR(64)      INIT(' ');
```

Abbildung 46. HLL Interface Area (PL/I Struktur)

6.13. HLL Interface Aufruftechnik

Ein User Programm, das einen ATO Dialog aufruft (CALL ATOHLL) muss entsprechend umgewandelt werden. Die folgenden Abbildungen zeigen die entsprechende Job Control zum Umwandeln eines User Programmes mit eingebautem ATO Dialog-Aufruf anhand eines Assembler Programmes. Beachten Sie bitte auch die weiteren Hinweise ab Seite 60.

```
// JOB HLL
// OPTION LOG,PARTDUMP
// LIBDEF *,SEARCH=(userlib.ATO410),CATALOG=(userlib.ATO410)
// OPTION CATAL
   ACTION CLEAR
   PHASE  TESTHLL,*
// EXEC ASSEMBLY
TESTHLL CSECT
        USING *,10
        BALR  10,0
        BCTR  10,0
        BCTR  10,0
        LA    13,SAVE
        MVC   HLLTO,DIALOG
        CALL  ATOHLL,(HLL)
        EOJ   RC=(15)
SAVE    DS    18F
DIALOG  DC    CL80'DIALOG=TEST'
        COPY  ASMHLL
        LTORG
        DROP  10
        END
/*
// EXEC LNKEDT
// LIBDEF *,SEARCH=(userlib.ATO410),CATALOG=(userlib.ATO410)
// OPTION CATAL
   ACTION CLEAR
   PHASE  TEST,*
// EXEC ASSEMBLY
        PROLOG ...,DIALOG=TEST,TIMEOUT=1440
        MAP KEY=CLEAR
        MAPEND
        MAP LASTMAP=YES,DATA='CESF LOGOFF'
        MAPEND
        EPILOG
        END
/*
// EXEC LNKEDT
// ASSGN SYS001,01E
// EXEC TESTHLL,SIZE=512K
/*
```

Abbildung 47. HLL Interface Aufruftechnik VSE

```
//jobname JOB ...
//STEP1   EXEC PROC=ATOGEN,DIALOG=TESTHLL
TESTHLL  CSECT
         USING *,10
         STM   14,12,12(13)
         LR    10,15
         LA    3,SAVE
         ST    3,8(13)
         ST    13,4(3)
         LR    3,13
         MVC   HLLTO,DIALOG
         CALL  ATOHLL,(HLL),VL
         L     13,4(13)
         L     14,12(13)
         LM    0,12,20(13)
         BR    14
SAVE     DS    18F
DIALOG   DC    CL80'DIALOG=TEST'
         COPY  ASMHLL
         LTORG
         DROP  10
```

```
         END
/*
//STEP2  EXEC PROC=ATOGEN,DIALOG=TEST
         PROLOG ...,DIALOG=TEST,TIMEOUT=1440
         MAP KEY=CLEAR
         MAPEND
         MAP LASTMAP=YES,DATA='CESF LOGOFF'
         MAPEND
         EPILOG
         END
/*
//STEP3  EXEC PGM=TESTHLL
//ATO... DD   SYSOUT=*
...
/*
```

Abbildung 48. HLL Interface Aufruftechnik MVS

6.14. HLL Interface Hinweise

Aufruf und Parameterübergabe

Ein ATO Dialog kann via ATOHLL nebst den verschiedenen Programmiersprachen auch durch Tools und Vendor-Utilities aufgerufen werden. Die genauen Anweisungen für den Dialog Aufruf entnehmen Sie bitte den entsprechenden Kapiteln, die Aufrufe von Assembler Modulen beschreiben.

ATOHLL kann von einem User Programm in Form von

- **o** Object Modulen

  oder

- **o** Phasen (VSE), Load Modulen (MVS)

aufgerufen werden.

Der Aufruf eines Object Modules ist eine weit verbreitete Methode und wird von allen bekannten Programmiersprachen unterstützt. Das Modul ATOHLL wird bei dieser Methode zur Linkzeit automatisch in das User Programm eingebunden. Der Entry- und der Loadpoint von ATOHLL ist identisch, daher müssen keine zusätzlichen Entry Statements für den Linkage Editor definiert werden.

Der Aufruf eines ATO Dialoges in einem User Programm als Phase oder Loadmodul wird vom entsprechenden Compiler mit entsprechenden Options unterstützt. Bei dieser Methode wird ATOHLL nicht in das User Programm eingebunden, sondern zur Ausführungszeit geladen.

Die Parameterübergabe erfolgt gemäss den Standard Linkage Konventionen und im ASM Standard Format.

Das ATOHLL Interface erwartet, dass die Länge, die im Feld HLLLENG definiert ist, in vollem Umfang zur Verfügung steht. Bei falschen Längenangaben können andernfalls Bereiche des User Programmes unbeabsichtigt überschrieben werden.

Das ATOHLL Interface setzt immer einen Return Code. Mit entsprechender Kodierung kann dieser im User Programm nach Rückkehr aus dem ATO Dialog überprüft werden. Bei Programmiersprachen, die keine Return Codes von Modulen unterstützen, steht dieser Return Code zusätzlich im Feld HLLRC zur Verfügung.

Storage Anforderungen

Durch den Aufruf eines ATO Dialoges aus einem User Programm via ATOHLL wird ATO in die gleiche Region (VSE) oder den gleichen Adressraum (MVS) wie das auszuführende User Programm geladen. Dadurch ergeben sich erhöhte Speicheranforderungen und zwar ca. 256 K für ATO und ca. 64 K für explizite Speicheranforderungen von ATO während der Ausführungszeit. Zusätzlich sind weitere implizite Speicheranfoderungen der aufgerufenen Schnittstellen, z.B. ATOM, DL/I, SQL etc. zu beachten.

Durch Angabe genügend grosser Regions oder Partitions mit der Angabe von

VSE: EXEC USERPGM,SIZE=nnnK (GETVIS für ATO und VTAM)
MVS: EXEC PGM=USERPGM,REGION=nnnK (Programmgrössen plus GETMAINS)

können Sie Programmabbrüchen infolge zu kleiner Partition/Region-Size vorbeugen.

Zusammenspiel mit Programmiersprachen

Verschiedene Programmiersprachen bieten Diagnose Hilfen, z.B. bei der Lokalisierung von Data Exceptions mittels SPIE/STXIT. Innerhalb von ATO wird kein SPIE/STXIT verwendet, da Program Checks höchtenfalls durch den User-Exit ATOEXI ausgelöst werden können. Dieser Umstand gewährleistet daher das volle Spektrum der Diagnose Hilfen für das User Programm in der gewählten Programmiersprache.

ATO arbeitet intern mit verschiedenen asynchronen Konstrukten und mit Multitasking. Mit der PROLOG Option TIMEOUT=1440 kann eine Task ausgeschaltet werden. Dies kann unter VSE erfoderlich sein, da nur eine beschränkte Anzahl von Task Levels unterstützt sind.

Entdeckt ATO nach dem Aufruf eines Dialoges durch ATOHLL einen Fehler, wird die Kontrolle mit einem entsprechenden Return Code an das User Programm zurückgegeben. Dies ermöglicht dem User Programm je nach Situation die entsprechenden Massnahmen durchzuführen wie z.B. Close von Files, Rewind von VSE Tapes oder unter DL/1 auf CBLTDLI zurückzukehren. Ein entsprechender Return Code wird auch gesetzt, wenn der aufgerufene ATO Dialog mit dem Befehl ABORT terminiert wird.

Beachten Sie, dass ein ATO Dialog wenn immer möglich mit der Sequenz MAP LASTMAP=YES abgeschlossen wird.

ATO kann innerhalb eines Batch-Programmes nur **einmal** gestartet werden.

JCL Definitionen

ATO benötigt für die Ein-/Ausgabe-Befehle GETRDR, PUTLOG und PUTPRT verschiedene Zuordnungen.

Unter MVS werden folgende DD-Statements verwendet:

```
//ATORDR  DD  ...
//ATOPRT  DD  ...
//ATOLOG  DD  ...

//SYSUDUMP DD ... (Diagnose Hilfe)
```

Beachten Sie, dass ATO keine DD-Zuordnungen mit //SYS... DD verwendet. Dies erlaubt Komponenten, die innerhalb eines User Programmes zur Anwendung kommen, Ein- bzw. Ausgaben auf SYSPRINT, SYSIN, SYSOUT etc. zuzuordnen, z.B. SORT Programme.

Funktion bei Aufruf aus einer Programmiersprache:

Die ganze HLL Interface Technik entfällt, da die höhere Programmiersprache bereits die Steuerung hat und ihr deshalb alle Areas, die für die Bedienung virtueller Bildschirme relevant sind, zur Verfügung stehen.

6.15. LOGTIME

Syntax:

	Befehl	*Parameter*
	LOGTIME	**SEC**=*sec*

sec **Gültige Werte:** 0 - 60

Dieser Parameter definiert die Wartezeit in Sekunden bei Verwendung von PROLOG LOGTERM.

Default: 3

Beschreibung:

Dieser Befehl dient zur Definition der Wartezeit zwischen den Ausgaben bei der Verwendung der PROLOG LOGTERM Funktion. LOGTIME kann an beliebiger Stelle im ATO-Dialog kodiert werden und ist sinnvoll, wenn der Wert des LOGTIME-Parameters im PROLOG-Befehl übersteuert werden soll. Mit diesem Befehl erreichen Sie eine grössere Effizienz bei der Kontrolle des ATO-Dialoges mittels der PROLOG LOGTERM Funktion, indem Sie z.B. bereits getestete Dialog-Sequenzen (z.B. Anmeldung etc.) schneller durchlaufen lassen und erst ab einer bestimmen Verarbeitungs-Sequenz ein grösseres Intervall setzen.

Beispiel:

```
       PROLOG ...,                              C
              LOGTERM=LOGON,                    C
              LOGTIME=0,                        C
              :

       :
       LOGTIME SEC=4
BILD50 MAP DATA='FIBU'
       MAPEND
```

Abbildung 49. Beispiel LOGTIME

6.16. LOOP

Syntax:

	Befehl	*Parameter*
	LOOP	**COUNT=***count*

count **Gültige Werte:** 1 - 100000

Dieser Parameter setzt den Wert des PROLOG LOOP-Parameters neu auf die gewünschte Zahl.

Default: 9000

Beschreibung:

Mit dem LOOP Befehl kann an beliebiger Stelle im Dialog der interne LOOP Counter auf einen neuen Wert gesetzt werden.

Beispiel:

```
       PROLOG ...,                                C
              LOGTERM=LOGON,                      C
              LOOP=50000,                         C
              :

       :
       LOOP COUNT=3000
BILD50 MAP DATA='FIBU'
       MAPEND
       SCAN DATA='HAUPT-MENU',FOUND=BILD60
       MAP KEY=PF3
       MAPEND
       MAP KEY=CLEAR
       MAPEND
       GOTO MARK=BILD50
BILD60 MAP
       FILL DATA='123456',LIN=07,COL=14
       MAPEND
       :
```

Abbildung 50. Beispiel LOOP

Funktion bei Aufruf aus einer Programmiersprache:

Da die Programmsteuerung durch eine höhere Programmiersprache erfolgt, wird auch dort geprüft, ob im Programmablauf Endlosschleifen entstehen.

6.17. MAP

Syntax:

	Befehl	*Parameter*
[*label*]	**MAP**	[**SESSID**='*sessid*' **,DATA**='*data*' ,**LIN**=*line* ,**COL**=*column* ,**RC**=*retcode* ,**LASTMAP**=*lastmap* ,**MDTAUTO**=*mdtauto* ,**TIMEOUT**=*timeout* ,**KEY**=*key* ,**RECEIVE**=*receive* ,**EMPTY**=*empty*]

Abbildung 51. MAP, Syntax

sessid **Gültige Werte:** Maximal 1 alphanumerisches Zeichen

sessid wechselt zu einer bestimmten Session. Dies ist notwendig, wenn mehrere Sessions parallel aktiv sind. *sessid* gilt für alle folgenden MAP-Befehle, die den Parameter nicht gesetzt haben.

Default: Momentan aktive Session

data **Gültige Werte:** Maximal 80 alphanumerische Zeichen

data kann einen konstanten Wert beinhalten oder beliebige alphanumerische Zeichen. Die Anzahl der Zeichen bestimmt die Länge des Feldes. Wird dem MAP-Befehl ein Label zugeordnet, kann aus dieses Feld mit TO-Parametern referenziert werden.

Default: Keiner

label **Gültige Werte:** Maximal 7 Zeichen

label kann als Sprungadresse für den GETRDR, GOTO oder SCAN Befehl verwendet werden oder als TO-Adresse zum Füllen von Daten des DATA Parameter.

Default: Keiner

line **Gültige Werte:** 01 - 43

Der Parameter definiert die Zeile, in welcher der Cursor im Eingabebild beim Betätigen der Enter-Taste oder PF-Taste positioniert ist.

Default: 01

column **Gültige Werte:** 01 - 132

Der Parameter definiert die Kolonne, in welcher der Cursor im Eingabebild beim Betätigen der Enter-Taste oder PF-Taste positioniert ist.

Default: 01

retcode **Gültige Werte:** 00, 04, 08, 16

Mit diesem Parameter kann ein Job-Control Returncode gesetzt werden. Siehe auch "Konventionen für Return-Codes" auf Seite 23.

Default: 00

lastmap **Gültige Werte:** NO, YES, PASS

Die Angabe von LASTMAP=YES informiert ATO, dass es sich um die letzte zu verarbeitende MAP-Anweisung handelt.

PASS wird verwendet, um von einer APPLID-Session zu einer anderen zu wechseln. Diese Funktionalität wird neu einfacher durch den Aufbau mehrerer Sessions gelöst. Vgl. dazu SESSBEG, SESSMOD/SESSSET, SESSEND.

Default: NO

mdtauto **Gültige Werte:** YES, NO

Bei Definition von MDTAUTO=NO werden nur Felder übertragen, die mit FILL oder MAPFLD übergeben werden.

Default: NO

timeout **Gültige Werte:** 1 - 60000

Mit der Angabe von TIMEOUT=*timeout* wird die Dauer in Sekunden spezifiziert, während der der Dialog maximal auf eine Antwort des TP-Monitors wartet. Ist dieser Wert überschritten, wird der Dialog abgebrochen.
Der Default-Wert wird mit dem PROLOG TIMEOUT-Parameter gesetzt.

Default: siehe PROLOG TIMEOUT-Parameter aus Seite 21.

SAP: Für ABAP/TF70/TF78/TK31 muss dieser Wert erhöht werden.

MVS: Für lang laufende Transaktionen ist der TIME-Parameter im JOB- oder EXEC-Statement anzupassen (z.B. TIME=1440) oder eine entsprechende Jobclass zur Ausführung des Dialoges zu wählen.

key **Gültige Werte:** ENT (Enter), CLEAR, PA1 - PA3, PF1 - PF24

Mit diesem Parameter wird die für diese Transaktion nötige Funtionstaste oder eine Enter-Eingabe bestimmt.

Default: ENT (Enter)

PA1 - PA3 sind nur im Transaktions-Modus unterstützt.

Wenn Sie die CLEAR-Taste zum Löschen eines Bildschirmes verwenden, so ist als nächstes die Befehlssequenz MAP DATA='*data*' zu verwenden, um Daten auf den unformatierten Bildschirm einzugeben.

CICS: PA1 ist für TCT-PRINTO / CSD-TERMINAL PRINTER() nicht unterstützt.

SAP: Zur besseren Übersicht verwenden Sie mit Vorteil den Funktionsablauf Nxxx auf Zeile/Kolonne 24/5 anstatt die vordefinierten Funktions-Tasten.

MVS: Die CLEAR Taste darf in TSO-Anwendungen nicht verwendet werden. PA1 wird im TSO verwendet, um die Ausgabe auf den Bildschirm nach einem TSO-Befehl zu unterbrechen.

receive **Gültige Werte:** YES, ONLY

Mit RECEIVE=ONLY kann ein zweites Bild vom TP-Monitor empfangen werden (siehe Beschreibung für PENDING ab Seite 80).

Default: YES

empty **Gültige Werte:** ALLOW

ATO erwartet nach jedem MAPEND eine Datenausgabe vom TP-Monitor. Mit EMPTY=ALLOW wird angezeigt, dass der Dialog fortgesetzt wird, auch wenn keine Datenausgabe vom TP-Monitor empfangen wurde.
Bei MAP KEY=CLEAR wird implizit EMPTY=ALLOW gesetzt.

SAP: Bei der Anwendung mit TF70 muss in der Schlaufe, welche mit PENDING die Bildschirme verarbeitet, EMPTY=ALLOW spezifiziert werden.

Default: keiner

Beschreibung:

Der Befehl MAP dient zum Positionieren des Cursors im Eingabebild und zur Definition der zu betätigenden Funktionstaste.

Zwischen MAP und MAPEND Befehlen dürfen nur FILL und MAPFLD Befehle stehen.

Beispiele:

```
CLEAR1  MAP KEY=CLEAR
        MAPEND
TRA1    MAP TIMEOUT=300,DATA='TRA1'
        MAPEND
        :
PFKEY   MAP LIN=07,COL=80,KEY=PF14
        MAPEND
        :
ENDE    MAP  KEY=PF3
        MAPEND
        :
        MAP  KEY=CLEAR
        MAPEND
        :
        MAP  LASTMAP=YES,RC=04,DATA='CESF LOGOFF'
        MAPEND
        :
```

Abbildung 52. Beispiel MAP

```
CLEAR1  MAP SESSID=A,KEY=CLEAR       Clear in Session A
        MAPEND
TRA1    MAP DATA='TRA1'              TRA1 an Session A senden
        MAPEND
        :
CLEAR2  MAP SESSID=B,KEY=CLEAR       Clear in Session B
        MAPEND
CEMT    MAP DATA='CEMT'              CEMT an Session B senden
        MAPEND
        :
PFKEY   MAP KEY=PF3                  PF3 in Session B
        MAPEND
        :
TRA1    MAP SESSID=A                 Mit Session A weiterfahren
        MAPKEYS DATA='...'
        MAPEND
```

Abbildung 53. Beispiel MAP mit mehreren Sessions

Funktion bei Aufruf aus einer Programmiersprache:

Die Bedienung virtueller Bildschirme ist aus Programmiersprachen heraus mit neuen Befehlen gelöst. Bitte vergleichen Sie dazu die Befehlsübersicht.

6.18. MAPEND

Syntax:

	Befehl	*Parameter*
	MAPEND	

Beschreibung:

Mit dem MAPEND Befehl wird eine MAP Anweisung abgeschlossen und eine Antwort des TP-Monitors erwartet. Der erhaltene Bildschirminhalt kann z.B. mit einem SCAN- oder MOVE-Befehl weiterverarbeitet werden.

Beispiel:

```
BILDN    MAP   DATA='MENU'
         MAPEND
         SCAN  FROM=SCREEN,...
         MOVE  FROM=SCREEN,...
         :
```

Abbildung 54. Beispiel MAPEND

Funktion bei Aufruf aus einer Programmiersprache:

Die Bedienung virtueller Bildschirme ist aus Programmiersprachen heraus mit neuen Befehlen gelöst. Bitte vergleichen Sie dazu die Befehlsübersicht.

6.19. MAPFLD

Syntax:

	Befehl	*Parameter*
[*label*]	**MAPFLD**	**DATA=**'*data*'

label **Gültige Werte:** Maximal 7 Zeichen

label kann mit einem TO-Parameter referenziert werden.

Default: Keiner

data **Gültige Werte:** Maximal 80 Zeichen

Dieser Parameter definiert den Wert, der einem Feld übergeben werden soll.

CICS: Der TCT-UCTRAN=YES / RDO-UCTRAN(YES) Parameter bewirkt die Umsetzung in Grossbuchstaben.

TSO: Innerhalb von TSO sind verschiedene Umsetzungen aktiv (z.B. Codepages im ISPF etc.).

Beschreibung:

MAPFLD übergibt den Wert im DATA=-Parameter einer MAP, wobei sich die Länge aus der Anzahl Zeichen zwischen den Hochkommas ergibt. Der MAPFLD-Wert kann z.B. mit den GETRDR-oder MOVE-Befehlen gefüllt werden. Die Beispiele in Abbildung 55 und 56 auf Seite 71 zeigen die Anwendung dieses Befehl.

MAPFLD-Befehle können nur zwischen MAP- und MAPEND-Befehlen kodiert werden. Generell kann mit diesem Befehl eine mit der DATA-Anweisung definiertes Feld mit Werten von variablem Inhalt unterteilt und/oder ergänzt werden.

Beispiele:

```
         :
         SCAN DATA='MAPPE'
         MOVE COL=6,TO=FNAME1
         :
BILDN    MAP
         FILL LIN=14,COL=07,DATA='ABSPIELUNG VON'
FNAME1   MAPFLD DATA='XXXXXXXX'
         MAPFLD DATA=' IST BEENDET'
         MAPEND
         :
EOF1     MARK
         :
```

Abbildung 55. Beispiel 1 MAPFLD

Dieses Beispiel trägt auf Zeile 14 ab Kolonne 7 den Wert "ABSPIELUNG VON " ein. Danach wird mit einem ersten MAPFLD-Befehl ein variabler Wert von 6 Zeichen angefügt. Da dem ersten MAPFLD-Befehl ein Label voransteht, kann das Feld mit einem TO= Parameter refernziert werden. Die zweite MAPFLD-Anweisung fügt zum Abschluss der Zeile den Wert " IST BEENDET" an, womit sich für diese Zeile der Wert "ABSPIELUNG VON XXXXXXXX IST BEENDET" ergibt.

```
         :
         MAP  KEY=CLEAR
         MAPEND
         MOVE TO=MAPPE,DATA='MAPPE1'
BILDN    MAP  DATA='SBDC S='
MAPPE    MAPFLD DATA='XXXXXX'
         MAPEND
         :
```

Abbildung 56. Beispiel 2 MAPFLD

Funktion bei Aufruf aus einer Programmiersprache:

Die Bedienung virtueller Bildschirme ist aus Programmiersprachen heraus mit neuen Befehlen gelöst. Bitte vergleichen Sie dazu die Befehlsübersicht.

6.20. MAPFILL

Syntax:

	Befehl	*Parameter*
	MAPFILL	**DATA=***data* **,LIN=***line* **,COL=***column* **,SESSID**=*sessid*

data **Gültige Werte:** Maximal 1 Zeile alphanumerische Zeichen (80 oder 132)

data kann einen konstanten Wert beinhalten oder beliebige alphanumerische Zeichen. Die Anzahl der Zeichen bestimmt die Länge des Feldes.

Default: Keiner

line **Gültige Werte:** 01 - 43

Der Parameter definiert die Zeile, in welcher die Eingabe erfolgt.

Default: 01

column **Gültige Werte:** 01 - 132

Der Parameter definiert die Spalte, in welcher die Eingabe beginnt.

Default: 01

sessid **Gültige Werte:** Maximal 8-stelliger Name

Mit diesem Parameter wird angegeben, in welcher Session die Dateneingabe erfolgen soll.

Default: Momentan aktive Session

Beschreibung:

Der Befehl MAPFILL dient der Dateneingabe auf formatierten Bildschirmen.

Beispiel:

```
CLEAR1  MAP KEY=CLEAR
        MAPEND
TRA1    MAP TIMEOUT=300,DATA='TRA1'
        MAPEND
        :
PFKEY   MAP LIN=07,COL=80,KEY=PF14
        MAPEND
        :
ENDE    MAP  KEY=PF3
        MAPEND
        :
        MAP  KEY=CLEAR
        MAPEND
        :
        MAP  LASTMAP=YES,RC=04,DATA='CESF LOGOFF'
        MAPEND
        :
```

Abbildung 57. Beispiel MAP

Funktion bei Aufruf aus einer Programmiersprache:

Die Bedienung virtueller Bildschirme ist aus Programmiersprachen heraus mit neuen Befehlen gelöst. Bitte vergleichen Sie dazu die Befehlsübersicht.

6.21. MAPKEYS

Syntax:

	Befehl	*Parameter*
[*label*]	**MAPKEYS**	**DATA=**'*data*'

label **Gültige Werte:** maximal 7 Zeichen

Wird dem MAPKEYS-Befehl ein Label zugeordnet, kann auf dieses Feld mit TO referenziert werden.

data **Gültige Werte:** maximal 80 Zeichen

Dieser Parameter enthält den Text, der an der definierten Position übergeben wird.

Eingebettet in den Text können auch Cursor-Steuerungstasten verwendet werden. Falls am Anfang nichts angegeben ist, beginnt der Text bei aktuellen Cursor-Position.

Mittels DATA='#VARn#' können Variablen aus dem User Exit ATOEXI übergeben werden. Vergleichen Sie dazu auch Kapitel "User Exit ATOEXI" auf Seite 129.

CICS: Die Verwendung von TCT-UCTRAN=YES Parameter bewirkt die Umsetzung in Grossbuchstaben.

TSO: Innerhalb von TSO sind verschiedene Umsetzungen aktiv (z.B. Codepages im ISPF etc.).

Beschreibung:

MAPKEYS dient zur Übergabe von Bildschirmwerten an ein Eingabefeld beginnend bei der aktuellen Cursor-Position. Die Eingabelänge ergibt sich aus der Anzahl Zeichen zwischen den Hochkommas abzüglich der durch @ gekennzeichneten Cursor-Steuerungstasten. Erstreckt sich der DATA-Parameter über 2 oder mehr Zeilen, muss auf Kolonne 72 ein Fortsetzungszeichen definiert werden. Die nächste Zeile muss in diesem Fall ab Kolonne 16 beginnen. Die Notation erfolgt gemäss Assembler-Konventionen. Vergleichen Sie dazu Beispiel 2 in Abbildung 57.

Beachten Sie, dass ein MAPKEYS-Befehl nur zwischen MAP- und MAPEND-Befehlen vorkommen darf.
Vergleichen Sie dazu die Beispiele in Abbildungen 57 und 58.

Anstelle des MAPKEYS-Befehls kann auch der Befehl FILL verwendet werden, wobei dort das Eingabefeld auf dem virtuellen Bildschirm durch Zielen- und Spalten-Angaben positioniert wird.

Bemerkung:

CICS: Die Eingabelänge wird von bestimmten Transaktionen wie z.B. CESN geprüft. Bei Eingaben von Transaktionscodes auf einem leeren Bildschirm wird mit Vorteil der Befehl MAP DATA='tran' verwendet. Dies entspricht einer Eingabe im Line-Modus.

Wird der Bildschirm mit MAP KEY=CLEAR gelöscht, so ist anschliessend MAP DATA='*data*' zu verwenden.

Beispiel:

```
BILDN   MAP
        MAPKEYS DATE='USERID@NPASSWORD'
        MAPEND
        :
```

Abbildung 58. Beispiel MAPKEYS

Im obigen Beispiel wird der String 'USERID' an der aktuellen Cursor-Position eingegeben, anschliessend folgt nach *newline* auf dem nächsten Eingabefeld einer anderen Zeile der String 'PASSWORD'.

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.22. MARK

Syntax:

	Befehl	*Parameter*
label	**MARK**	

label **Gültige Werte:** Maximal 7 Zeichen

label definiert eine Sprungmarke und ist eine **zwingende** Eingabe.

Default: Keiner

Beschreibung:

MARK dient zur Definition einer Sprungmarke. Diese kann in den Befehlen GETRDR, GOTO, SCAN, SCANB und PENDING verwendet werden.

Intern wird von MARK ein Loop-Counter geführt. Dieser dient zur Verhinderung endloser Schlaufen und kann im PROLOG LOOP-Parameter auf einen Maximalwert gesetzt werden.

Beispiel:

```
        GETRDR EOF=EOF1
        :
        SCAN DATA='HAUPT-MENU',NFOUND=F1
        :
        GOTO MARK=ENDE
        :
EOF1    MARK
        :
F1      MARK
        :
ENDE    MARK
        :
```

Abbildung 59. Beispiel MARK

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.23. MOVE

Syntax:

	Befehl	*Parameter*
	MOVE	[**DATA=**'*data*' | **FROM=***fromfield* ,**COL=***column* ,**LIN=***line* ,**TO=***tofield* ,**TOCOL=***tocolumn* ,**TOLEN=***tolength*]

data **Gültige Werte:** Maximal 80 Zeichen

Dieser Parameter die Daten, die einem Feld übergeben werden soll.

Die übertragene Länge der Daten wird durch die Anzahl Zeichen zwischen den Hochkommas bestimmt.

fromfield **Gültiger Wert:** WORK1 - 9, SCREEN, SPACES oder ein Label

Dieser Feldname definiert einen Bereich, aus dem Werte übertragen werden.

Default: WORK1

column **Gültige Werte:** 01 - 132

Dieser Parameter definiert die Kolonne von WORK1 - 9 und SCREEN, ab welcher der Wert übergeben werden soll.

Default: 01

line **Gültige Werte:** 01 - 43

Dieser Parameter definiert die Zeile, ab welcher der Wert von **SCREEN** übergeben werden soll.

Default: 01

tofield **Gültiger Wert:** Maximal 7-stelliger Feldname, WORK1 - 9 oder ein Label

Dieser Feldname definiert einen Bereich, in den Werte übertragen werden.

Default: WORK1

tocolumn **Gültige Werte:** 01 - 132

Dieser Wert definiert die Kolonne von WORK1 - 9, in welche übertragen werden soll.

Default: 1

tolength **Gültige Werte:** 01 - 132

Dieser Wert definiert die Länge der Daten in WORK1 - 9, die übertragen werden sollen.

Default: 132

Beschreibung:

MOVE dient zum Übertragen verschiedenster Bereiche. Bei WORK1 - 9 / SCREEN können zusätzlich die Parameter TOLEN, TOCOL, LIN und COL definiert werden.

Beispiele:

```
         :
BILD1    MAP  DATA='TRA1'
         MAPEND
         SCAN DATA='MAPPE',NFOUND=NF1
         MOVE FROM=SCREEN,TO=FNAME1
         :
         MOVE FROM=SCREEN,COL=8,LIN=3,TO=FNUMM1
         :
BILD2    MAP
         FILL LIN=14,COL=07,DATA='ABSPIELUNG VON'
FNAME1   MAPFLD DATA='...MAPPE...'
         MAPFLD DATA=' IST BEENDET, NR='
FNUMM1   MAPFLD DATA='000000'
         MAPEND
         :
NF1      MARK
         :
```

Abbildung 60. Beispiel 1 MOVE

```
        :
B10     MAP  DATA='FIBU'
        MAPEND
        :
L20     MARK
        GETRDR TO=WORK2,EOF=END1
        MOVE FROM=WORK2,COL=1,TO=TEXT
        MOVE FROM=WORK2,COL=13,TO=MENGE
        MOVE FROM=WORK2,COL=21,TO=ART
        :
B20     MAP
TEXT    FILL LINE=10,COL=14,DATA='XXXXXXXXXXX'
MENGE   FILL LINE=11,COL=20,DATA='XXXXX.XX'
ART     FILL LINE=12,COL=10,DATA='XX'
        MAPEND
        :
        GOTO MARK=L20
END1    MARK
        :
```

Abbildung 61. Beispiel 2 MOVE

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.24. PENDING

Syntax:

	Befehl	*Parameter*
	PENDING	[**FOUND=***label* ,**NFOUND=***label* ,**TIMEOUT=***sec*]

label **Gültige Werte:** Maximal 7 Stellen

label ist eine mit MARK *label*-MAP definierte Sprungmarke für den FOUND- oder NFOUND-Fall .

sec **Gültige Werte:** 1 - 60000

Siehe TIMEOUT-Parameter im Kapitel "MAP" auf Seite 63.

Default: 60

Als Alternative zu PENDING kann auch der Befehl MAP RECEIVE=ONLY verwendet werden, falls eine TP-Monitor-Transaktion **immer an derselben Stelle** eine zweite Bildschirm-Ausgabe schickt.

CICS: PENDING wird zum Empfangen von ausserordentlichen Meldungen (z.B. CMSG) verwendet.

SAP: PENDING kann z.B. zum Abspielen von Mappen via SBDC oder TF70 / ABAP verwendet werden.

Beschreibung:

ATO arbeitet streng nach dem Dialog-Takt von Ein-/Ausgaben, welche durch die Befehle MAP / MAPEND verarbeitet werden.

Ausser bei der ersten Transaktion bestimmt immer ATO, welche als nächstes verarbeitet werden soll. Eine Ausnahme bilden die Befehle PENDING und MAP RECEIVE=ONLY.

Startet der TP-Monitor eine Transaktion, kollidiert diese mit dem Transaktionsstart von MAP / MAPEND und wird von ATO zurückgewiesen. TP-Monitoren wie CICS versuchen die pendente Transaktion bei jedem Transaktionsende erneut zu starten, was zu einer wiederholten Kollision und Zurückweisung durch ATO führt.

Ist nach einer Zurückweisung durch ATO im TP-Monitor eine Transaktion pendent, kann diese mit dem PENDING-Befehl ausgelöst werden, womit der TP-Monitor die "pendente" Transaktion startet.

Trifft die PENDING-FOUND Bedingung zu, hat der TP-Monitor die Transaktion gestartet. ATO verhält sich dabei wie nach einem MAPEND, dass heisst, der Bildschirm ist für die weitere Verarbeitung bereit.
Ist hingegen die PENDING-NFOUND Bedingung erfüllt, reagiert ATO wie nach einem GOTO, indem die Verarbeitung an der im NFOUND Parameter definierten Stelle fortgesetzt wird.

CICS: Wird ein Dialog abgebrochen, liegt es in der Verantwortlichkeit von CICS zu entscheiden, was mit der pendenten Transaktion geschehen soll. CICS startet eine solche Transaktion jeweils unmittelbar nach dem Wiederaufbau des Dialoges mit demselben logischen Terminal, anstatt der erwarteten "Good Morning"-Transaktion. Dies kann zu einer Verletzung des Security- Levels führen, da der ATO-Bildschirm noch nicht angemeldet ist (z.B. via CESN).

SAP; Bei lang laufenden Transaktionen, was z.B. beim Abspielen von Mappen via SBDC, ABAP oder TF70 der Fall sein kann, wird die "Beendet"-Meldung durch eine CICS Start-Transaktion ausgelöst. Da diese Meldung unter Umständen erst nach Stunden eintrifft, muss die Kollision ausgelöst werden. Siehe dazu das Beispiel in Abbildung 62 auf Seite 81.

Bemerkung:

PENDING dient hauptsächlich dazu, Meldungen und Transaktionen von TP-Monitoren ausserhalb des Dialog-Taktes abzufangen. Diese können bei CICS-Anwendungen z.B. durch EXEC CICS START TRANSID oder durch die CMSG-Transaktion gestartet werden.

Da diese "ausstehenden" Transaktionen immer auf ein bestimmtes ATO-Terminal bezogen sind, wird pro ATO-Anwendung mit Vorteil ein reserviertes ATO-Terminal verwendet, um damit Konflikte mit vorgängig durchgeführten Dialogen auszuschliessen.
Für SAP-Anwendungen empfiehlt es sich zudem, pro ATO-Terminal einen reservierten SAP-User zu verwenden.

Beispiele:

```
          :
REDO1     MAP          * Kollision
          MAPEND
          SCAN DATA='BEENDET',FOUND=END1
          PENDING FOUND=END1
          SLEEP
          GOTO MARK=REDO1
END1      MARK
          :
```

Abbildung 62. Beispiel 1 PENDING CICS-SAP

```
          :
          MAP DATA='TRA1'
          MAPEND
          SCAN DATA='MELDUNG VON TRA1'
          :
          PENDING NFOUND=NTRA2
          SCAN DATA='MELDUNG VON TRA2'
          :
NTRA2     MARK
          :
ENDE      MARK
          MAP KEY=PF3
          MAPEND
          PENDING
          MAP KEY=CLEAR
          MAPEND
          MAP LASTMAP=YES,DATA='CESF LOGOFF'
          MAPEND
          :
```

Abbildung 63. Beispiel 2 PENDING

```
          :
B1        MAP  DATA='CEMT INQ TAS'
          MAPEND
B2        MAP KEY=ENT
          MAPEND
*********************************************
B3        MAP KEY=ENT
          MAPEND
B4        MAP KEY=PF3
          MAPEND
*********************************************
* Bei der nächsten Transaktion tritt       *
* eine Kollision mit CMSG auf              *
*********************************************
B5        MAP  DATA='CMSG'
          MAPEND
*********************************************
* Nun ist die Kollision eingetreten        *
*********************************************
          PENDING
*********************************************
* Der neue Bildschirminhalt kann verarbeitet*
* werden                                   *
*********************************************
          SCAN DATA='WELCHE MELDUNG'
          :
```

Abbildung 64. Beispiel 3 PENDING CICS

6.25. PERFORM

Syntax:

	Befehl	*Parameter*
	PERFORM	**PROC=***procname*

procname **Gültige Werte:** Maximal 7 alphanumerische Zeichen

Dieser Parameter bezeichnet den Namen der auszuführenden Prozedur.

Default: keiner

Beschreibung:

Der Befehl PERFORM dient zur Ausführung von Prozeduren. Mit PERFORM können Befehlssequenzen, die in der Prozedur *procname* kodiert sind, ausgeführt werden. Die aufgerufene Prozedur muss mit dem Befehl PROC definiert sein.

Weitere Informationen entnehmen Sie bitte dem PROC / PROCEND Befehl auf Seite 85 und 86.

Beispiel:

```
          PROLOG DIALOG=SAMPLE1,                  C
               NETNAME=NETATO1,                   C
               APPLID=DBDCCICS
          :
          MAP DATA='TRA1'
          MAPEND
          PERFORM PROC=READ
          :
          PERFORM PROC=CHECK
          SCAN DATA='ERROR',FROM=WORK2,           C
               NFOUND=CONT20
          PUTPRT DATA='LINE NOT INSERTED'
          PERFORM PROC=ERR08
CONT20    MARK
          :
          MAP LASTMAP=YES,...
          MAPEND
*************************************************
* PROZEDUR READ                                 *
*************************************************
READ      PROC
READ10    MARK
          GETRDR TO=LINE,EOF=EOF99
MAP10     MAP
LINE      FILL DATA='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXC
               XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
          MAPEND
          GOTO MARK=READ10
EOF99     MARK
          PROCEND
*************************************************
* PROZEDUR FEHLERHANDLING                       *
*************************************************
ERR08     PROC
          MAP  LASTMAP=YES,RC=08,DATA='CESF LOGOFF'
          MAPEND
          PROCEND
*************************************************
* PROCEDURE CHECK                               *
*************************************************
CHECK     PROC
          MOVE TO=WORK2,FROM=SPACES
          SCAN DATA='LINE INSERTED',COL=05,       C
               LIN=24,FOUND=CHECK05
          MOVE TO=WORK2,DATA='ERROR'
CHECK05   MARK
          PROCEND
*************************************************
* END OF DIALOG                                 *
*************************************************
          EPILOG
          END
```

Abbildung 65. Beispiel PERFORM und PROC

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.26. PROC

Syntax:

	Befehl	*Parameter*
procname	**PROC**	

procname **Gültige Werte:** Maximal 7 alphanumerische Zeichen

procname definiert den Namen der Prozedur und ist eine **zwingende** Eingabe. Die Schachtelung beträgt maximal 32 Stufen.

Default: keiner

Beschreibung:

Der Befehl PROC kennzeichnet den Anfang einer Prozedur. Der Inhalt einer solchen Prozedur sind normale Dialog-Befehlssequenzen, die mit PERFORM PROC=*procname* aufgerufen werden können. Die Prozedur endet mit dem PROCEND Befehl. Innerhalb von PROC / PROCEND können weitere PROC / PROCEND- und PERFORM-Befehle verwendet werden. Eine Prozedur kann auch sequentiell ohne vorgängigen Aufruf durch PERFORM durchlaufen werden.

Wichtig:

Innerhalb einer Prozedur dürfen keine Labels referenziert oder angesprungen werden, die ausserhalb PROC / PROCEND liegen.

Wir empfehlen Ihnen, die Proceduren zur besseren Übersicht nach der MAP-LASTMAP=YES Definition zu plazieren.

Ein Beispiel für PROC finden Sie in Abbildung 65 auf Seite 84.

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.27. PROCEND

Syntax:

	Befehl	*Parameter*
[*label*]	**PROCEND**	

label **Gültige Werte:** Maximal 7 alphanumerische Zeichen

label definiert eine Sprungmarke, auf die verzweigt werden kann.

Default: keiner

Beschreibung:

Mit dem PROCEND Befehl wird das Ende einer Prozedur markiert. Die Kontrolle geht zurück an die Anweisung, die dem aufrufenden PERFORM Befehl folgt. Wird eine Befehlssequenz PROC / PROCEND sequentiell, d.h. ohne vorgängigen Aufruf durch PERFORM durchlaufen, erfolgt bei PROCEND kein Rücksprung.

Ein Beispiel für PROC finden Sie in Abbildung 65 auf Seite 84.

Funktion bei Aufruf aus einer Programmiersprache:

Wird durch die Programmiersprache geregelt. Bitte vergleichen Sie dazu die ensprechende Literatur.

6.28. PROLOG

Syntax:

	Befehl	*Parameter*
	PROLOG	**DIALOG=***name* ,**NETNAME=***netname* ,**APPLID=***applid* ,**APPLTRY,=***appltry* ,**LOGTERM=***logterm* ,**LOGTIME=***logtime* ,**LOGTRY=***logtry* ,**LOGAID=***logaid* ,**LOOP=***loop* ,**LINEOV=***lineov* ,**MLOG=***mlog* ,**MDTAUTO=***mdtauto* ,**TRACE=***trace* ,**TIMEOUT=***timeout* ,**SESSMAX=***sessmax* ,**SUPPORT=***support* ,**LOGMODE=***logmode* ,**LOGTMOD=***logtmod*

Abbildung 66. PROLOG, Syntax

name **Gültige Werte:** Maximal 8 alphanumerische Zeichen

Dieser Parameter bezeichnet den Namen des auszuführenden Dialoges.

Default: ATODLOG

netname **Gültige Werte:** Maximal 8-stelliger Terminalname gemäss Netzwerk-Konventionen

CICS: Dieser Parameter bezeichnet einen gültigen im CICS definierten oder im AUTOINSTALL zugelassenen Netzwerknamen.

Default: NETATO1

applid **Gültige Werte:** APPLID Name gemäss Netzwerk-Konventionen

APPLID für eine ATO Session.

CICS: Dieser Parameter bezeichnet den in der SIT des entsprechenden CICS eingetragenen Applikationsnamen**.** Dieser ist auch mit der Transaktion CEMT INQ TAS am unteren rechten Bildrand ersichtlich.

Default: DBDCCICS

appltry **Gültige Werte:** 1 - 6000

appltry ist die Anzahl der Versuche, die ATO für die Herstellung einer Verbindung zur Applikation in Parameter APPLID durchführt.

Die Angabe von **APPLTRY=1** kann verwendet werden, um einen Dialog nur dann auszuführen, wenn der TP-Monitor aktiv und verfügbar ist.

Default: 6000

logterm **Gültige Werte:** Maximal 8 alphanumerische Zeichen. Gültiger Netzname gemäss Netzwerk-Konventionen, NO oder LOGON.

Dieser Parameter definiert den Netzwerknamen eines Bildschirms für die Überwachung und Schritt-für-Schritt-Anzeige dieses Dialoges. Ist diese Anzeige nicht erwünscht, so ist LOGTERM=NO zu spezifizieren.

Bei Angabe von LOGTERM=LOGON kann man sich dynamisch in den LOGTERM einschalten durch Eingabe von LOGON APPLID(*netname*). ATO wartet dabei mit der Verarbeitung , bis ein LOGON APPLID(*netname*) erfolgt.

Bemerkung:

LOGTERM ist auch für Session-Manager Produkte unterstützt.

Die Verwendung von LOGON APPLID(...) entspricht dem Default USSCMD-FORMAT=PL1 in der USSTAB Definition. Bei Angabe von FORMAT=BAL ist LOGON APPLID=... zu verwenden. Die Eingabe erfolgt bei USSTAB Message 10.

Default: NO

logtime **Gültige Werte:** 0 - 60

logtime definiert eine Zeitverzögerung in Sekunden mit LOGTERM. Während dieser Zeit wartet ATO mit der Ausgabe des nächsten Bildes.
Mit dem Befehl LOGTIME kann der Wert im Dialog neu gesetzt und übersteuert werden (Siehe Befehl LOGTIME auf Seite 63).

Default: 2

logtry **Gültige Werte:** 0 - 6000

logtry definiert die Anzahl der Versuche, um eine Verbindung zum LOGTERM Bildschirm herzustellen. Bei Angabe von **LOGTRY=1** wird der LOGTERM Bildschirm nur verwendet, wenn dieser auch verfügbar ist bzw. dieser im VTAM definiert ist. Ansonsten wird die Verarbeitung ohne LOGTERM fortgesetzt.

Default: 6000

logaid **Gültige Werte:** YES oder NO.

Wenn dieser Parameter auf YES steht, wird beim Verfolgen der Session auf dem LOGTERM unten rechts der Key angezeigt, der gedrückt wurde. Damit wird das Finden von Fehlern mit Hilfe von LOGTERM vereinfacht.

Steht LOGTERM auf NO, hat der Wert von LOGAID keinen Einfluss.

Default: NO

loop **Gültige Werte:** 1 - 100000

loop definiert den Maximalwert, den der Loop-Counter erreichen kann. Beim Erreichen dieses Werts wird der Dialog abgebrochen.

Default: 9000

lineov **Gültige Werte:** 1 - 99999

Mit *lineov* wird die Anzahl Zeilen pro Seite bestimmt.

Default: 55

mlog **Gültige Werte:** YES oder NO.

Wenn dieser Parameter auf YES steht, werden alle durchlaufenen Macros und Labels auf ATOPRT geschrieben. Damit kann im Fehlerfall einfach nachvollzogen werden, wo der Dialog genau durchlauft.

Default: NO

mdtauto **Gültige Werte:** YES, NO

Bei Angabe von MDTAUTO=YES werden für MDT-Bildschirmfelder automatisch FILL-Befehle generiert (z.B. CICS BMS-FSET Felder). Damit können auch Felder übertragen werden, die für den Benutzer auf dem Bildschirm unsichtbar (Dark) bleiben und nur der Dialogsteuerung dienen.

Bei Definition von MDTAUTO=NO werden nur Felder übertragen, die mit MAPFILL eingefügt oder verändert wurden.

Default: YES

timeout **Gültige Werte:** 0 - 60000

Dieser Parameter definiert die maximale Zeit in Sekunden, während der ATO auf eine Antwort wartet. Dieser Parameter setzt den Default für MAP- und PENDING-TIMEOUT. Der Default-Wert genügt für die meisten Anwendungen.

Mit der Angabe von TIMEOUT=1440 wird die Zeitüberschreitungs-Kontrolle ausgeschaltet. TIMEOUT=1440 sollte nur für langlaufende Dialoge verwendet werden.

Default: 120

trace **Gültige Werte:** NO, YES

Dieser Parameter dient zum Ein- bzw. Ausschalten des ATOLOGs. Wenn TRACE=YES gesetzt ist, werden alle Bildschirme angezeigt, wie sie im Dialog durchlaufen werden.

Default: YES

sessmax **Gültige Werte:** 1 - 9

Dieser Parameter gibt die Anzahl Sessions an, die in einem Dialog gleichzeitig aktiv sein können.

Default: 1

support **Gültige Werte:** NO, YES

Dieser Parameter dient der Diagnose-Hilfe. Bei Verwendung von SUPPORT=YES werden detaillierte Dialogauswertungen auf den ATOLOG geschrieben. Dabei werden auch die Werte aus dem User Exit ATOEXI ausgegeben.

Default: NO

VSE: Bei SUPPORT=YES muss der JCL-Parameter //OPTION LOG,PARTDUMP definiert sein.

MVS: Bei SUPPORT=YES muss der JCL-Parameter MSGLEVEL=(1,1), sowie //SYSUDUMP definiert sein.

logmode **Gültige Werte:** LOGMODE Name gemäss VTAM Definition

Mit diesem Parameter wird spezifiziert, welcher LOGMODE für den virtuellen Bildschrim gelten soll.

Es können alle Bildschirm-Modelle (Modell 2, 3, 4 oder 5) verwendet werden.
Default: D4A32782

logtmod **Gültige Werte:** LOGMODE Name gemäss VTAM Definition

Mit diesem Parameter wird spezifiziert, welcher LOGMODE für den LOGTERM-Bildschrim gelten soll. In der Regel sollte derselbe LOGMODE wie beim *logmode* Parameter angegeben werden.

Es können alle Bildschirm-Modelle (Modell 2, 3, 4 oder 5) verwendet werden.

Default: D4A32782

Beschreibung:

Der PROLOG Befehl bestimmt den Beginn eines ATO Dialoges und ist damit **zwingend** die erste zu verarbeitende ATO-Anweisung. Durch entsprechende Parameter wird mitgeteilt, welcher TP-Monitor als Default gilt und mit welchem logischen Terminal die Verarbeitung erfolgen soll. Weiter kann mit den jeweiligen Parametern ein detailliertes LOG oder ein zum Dialog parallel laufendes LOG-Terminal angefordert werden, welches ebenfalls zur Diagnose im Support-Center des Lizenzgebers verwendet wird.

Das Zusammenspiel der PROLOG-Definitionen mit denjenigen der Applikationen und von VTAM ist im "*Installation Manual*" beschrieben.

Bemerkungen:

PROLOG enthält implizit ein SESSSET und SESSBEG für SESSION A. Die mit PROLOG aufgebaute Session muss nicht mit SESSEND beendet werden.

Der PROLOG TIMEOUT-Parameter bestimmt zusätzlich die Einstellung auf die zu erwartende Antwortzeit.

Wird bei einer MAP-Eingabe eine längere Antwortzeit erwartet als diejenige des PROLOG TIMEOUT-Parameters, muss der MAP resp. PENDING TIMEOUT-Parameter entsprechend gesetzt werden. Damit wird der Zeitwert für diese Eingabe lokal definiert.

Der PROLOG TIMEOUT-Wert sollte nicht unnötig erhöht werden, da ATO sonst keine internen logische Fehler erkennen kann. Für einzelne Eingaben mit erwartungsgemäss längerer Antwortzeit ist der MAP TIMEOUT-Parameter zu verwenden.

Der PROLOG APPLTRY-Parameter wird verwendet, um einen Dialog nur dann durchzuführen, wenn der TP-Monitor verfügbar ist. Zu diesem Zweck definieren Sie APPLTRY=1.

Beispiel:

```
PROLOG DIALOG=SAMPLE1,                C
     NETNAME=NETATO1,                 C
     APPLID=DBDCCICS,                 C
     LOGTERM=LOGON
:
EPILOG
END
```

Abbildung 67. Beispiel PROLOG

Funktion bei Aufruf aus einer Programmiersprache

ATO_INITIALIZE
ATO_SET_SESSION_PARAMTERS

6.29. PUTLOG

Syntax:

	Befehl	*Parameter*
	PUTLOG	[**DATA**='*data*' | ,**FROM**=*field*]

data **Gültige Werte:** Maximal 64 Stellen alphanumerisch.

Der Text *data* wird auf ATOLOG ausgegeben. Wird dieser Parameter ausgelassen, wird der Inhalt des mit dem FROM-Parameter definierten Feld oder dessen Default ausgegeben.

Default: Keiner

field **Gültige Werte:** WORK1 - 9 oder vordefiniertes Feld (z.B. mit DCL, MAP, FILL, SCAN(B), MAPFLD etc.)

Der Inhalt aus *field* wird auf ATOLOG ausgegeben.

Default: WORK1

Beschreibung:

PUTLOG schreibt entweder den mitgegebenen Text im DATA-Parameter oder den Inhalt des mit FROM angegebenen Feldes auf ATOLOG. Es kann nur EINER der beiden Parameter definiert werden. Wird kein Parameter mitgegeben, ist der Default PUTLOG FROM=WORK1

Bemerkung:

VSE: PUTLOG schreibt auf SYSLST.

MVS: PUTLOG schreibt auf //ATOLOG DD SYSOUT=*.
ATOLOG hat eine DCB-Definition von DCB=(LRECL=121,BLKSIZE=121,RECFM=FBA).

Um Meldungen abzusetzen, die nur der Kontrolle eines Dialogs dienen, d.h. reine Informationshinweise, sollte der Befehl PUTPRT verwendet werden. Vergleichen Sie dazu Kapitel "PUTPRT" auf Seite 95.

Beispiel:

```
MSG01   DCL DATA='FEHLER: KEIN HAUPT-MENU'
        :
BILD1   MAP  DATA='MENU'
        MAPEND
        SCAN DATA='HAUPT-MENU',NFOUND=FEHLER
        :
FEHLER  MARK
        PUTLOG FROM=MSG01
        :
```

Abbildung 68. Beispiel PUTLOG

Funktion bei Aufruf aus einer Programmiersprache

File I/O wird mit Befehlen der Programmiersprache durchgeführt. ATO_PUTLOG kann ebenfalls verwendet werden.

6.30. PUTPRT

Syntax:

	Befehl	*Parameter*
[*label*]	**PUTPRT**	[**DATA**='*data*' | ,**FROM**=*field* ,**TITLE**=*titlenr*]

label **Gültige Werte:** Maximal 7 alphanumerische Zeichen.

label kann als Sprungadresse in einem GOTO verwendet werden. Es ist jedoch nicht möglich, *label* mit TO / FROM zu referenzieren.

data **Gültige Werte:** Maximal 64 Stellen alphanumerisch.

Siehe PUTLOG auf Seite 93.

Default: Keiner

field **Gültige Werte:** WORK1 - 9 oder vordefiniertes Feld (z.B. mit DCL, MAP, FILL, SCAN(B), MAPFLD etc.)

Der Inhalt aus *field* wird auf ATOPRT ausgegeben.

Default: WORK1

titlenr **Gültige Werte:** 1, 2, 3

Dieser Parameter definiert Titelzeilen mit dem Text des DATA-Parameters. In diesem Fall ist eine Titelzeile eindeutig mit dem TITLE-Parameter identifiziert. Handelt es sich beim eingegebenen Text nicht um eine Titelzeile, wird dieser Parameter ausgelassen.

Default: blank

Beschreibung:

PUTPRT schreibt entweder den mitgegebenen Text im DATA-Parameter oder den Inhalt des mit FROM definierten Feldes auf ATOPRT. Dies erleichtert die Kontrolle des Outputs, z.B. für das RZ-

Personal. Es kann nur **einer** der beiden Parameter definiert werden. Wird kein Parameter mitgegeben, ist der Default PUTPRT FROM=WORK1.

Bemerkung:

VSE: PUTPRT schreibt auf SYS001 und kann z.B. mit // ASSGN SYS001,01E,TEMP auf 01E gesetzt werden.
Der virtuelle Printer ist mit POWER-Statements zu referenzieren.

MVS: PUTPRT schreibt auf //ATOPRT DD SYSOUT=*.
ATOPRT hat eine DCB-Definition von
DCB=(LRECL=121,BLKSIZE=121,RECFM=FBA).

Beispiel:

```
      :
      PUTPRT TITLE=1,DATA='T I T E L Z E I L E'
      PUTPRT TITLE=2,DATA='------------------'
      PUTPRT TITLE=3,DATA='                  '
      :
      PUTPRT DATA='ANMELDUNG ERFOLGREICH'
      :
MSG01 DCL DATA='MAPPEN NICHT GEFUNDEN'
      PUTPRT FROM=MSG01
      :
```

Abbildung 69. Beispiel PUTPRT

Funktion bei Aufruf aus einer Programmiersprache

File I/O wird mit Befehlen der Programmiersprache durchgeführt.

6.31. PUTWTO

Syntax:

	Befehl	*Parameter*
	PUTWTO	[**DATA=**'*data*' | ,**FROM=***field*]

data **Gültige Werte:** Maximal 64 Stellen alphanumerisch.

Siehe PUTLOG auf Seite 93.

Default: keiner

field **Gültige Werte:** WORK1 - 9 oder vordefiniertes Feld (z.B. mit DCL, MAP, FILL, SCAN(B), MAPFLD etc.)

Der Inhalt aus *field* wird auf die Konsole ausgegeben.

Default: WORK1

Beschreibung:

PUTWTO schreibt entweder den mitgegebenen Text im DATA-Parameter oder den Inhalt des mit FROM definierten Feldes auf die Konsole. Es kann nur EINER der beiden Parameter definiert werden. Wird kein Parameter mitgegeben, ist der Default PUTWTO FROM=WORK1.

Beispiel:

```
MSG01     DCL DATA='ATO DIALOG BEENDET'
          :
          PUTWTO DATA='SIGN-ON FEHLER'
          PUTWTO FROM=MSG01
          ABORT RC=08
          :
```

Abbildung 70. Beispiel PUTWTO

Funktion bei Aufruf aus einer Programmiersprache

Die Funktion wird mit Befehlen der Programmiersprache durchgeführt. ATO_PUTWTO kann ebenfalls verwendet werden.

6.32. SCAN

Syntax:

	Befehl	*Parameter*
[*label*]	**SCAN**	**DATA**='*data*' [,**FROM**=*from* ,**TO**=*to* ,**LIN**=*line* ,**COL**=*col* ,**ENDLIN**=*endlin* ,**ENDCOL**=*endcol* ,**TOFFSET**=*toffset* ,**FOUND**=*mark* ,**NFOUND**=*mark*]

label **Gültige Werte:** Maximal 7 alphanumerische Zeichen.

label kann mit TO / FROM referenziert werden (z.B. mit MOVE).

data **Gültige Werte:** Maximal 80 Zeichen in Grossbuchstaben.

Der definierte Wert *data* wird auf dem empfangenen Bildschirminhalt gesucht.

Default: keiner

from **Gültige Werte:** SCREEN, WORK1 - 9 oder vordefiniertes Feld

from bezeichnet den Bereich, in dem der String im DATA Parameter gesucht wird.

Default: SCREEN (Bildschirm)

to **Gültige Werte:** WORK1 - 9 oder vordefiniertes Feld

to bezeichnet einen Bereich, in den die nächsten 80 Zeichen ab dem gefundenen String im DATA Parameter gestellt werden oder die Daten, die innerhalb der mit ENDLIN und ENDCOL definierten Position liegen. Bei WORK1 - 9 werden 80 Zeichen übertragen.

Default: WORK1

lin **Gültige Werte:** 01 - 43

lin definiert die Zeilenposition bei einem SCAN auf den SCREEN.

Default: 01

col **Gültige Werte:** 01 - 132

col definiert die Kolonnenposition, ab welcher der SCAN auf SCREEN oder WORK1 - 9 beginnen soll.

Default: 01

endlin **Gültige Werte:** 01 - 43

Mit *endlin* kann eine Endposition einer Zeile für den SCAN auf SCREEN definiert werden. Der String im DATA Parameter muss bei Angabe von ENDLIN innerhalb der Zeilen liegen, die mit LIN und ENDLIN definiert wurden. Trifft dies nicht zu, wird die NFOUND-Bedingung gesetzt. Der Wert *endlin* muss **grösser** oder **gleich** dem Wert in LIN sein.

Ist LIN spezifiziert und für ENDLIN nichts angegeben, wird nur auf der Zeile *lin* gesucht.

Default: 01

endcol **Gültige Werte:** 01 - 132

Mit *endcol* kann eine Endposition einer Kolonne für den SCAN auf SCREEN oder WORK1 - 9 definiert werden. Der String im DATA Parameter muss bei Angabe von ENDCOL innerhalb einer Kolonne liegen, die zwischen COL und ENDCOL liegt. Trifft dies nicht zu, wird die NFOUND-Bedingung gesetzt. Der Wert muss **grösser** oder **gleich** dem Wert in COL sein. Falls *col* und *endcol* den gleichen Wert haben, wird der String ab *col* in der Länge des Strings gesucht~~auf der ganzen Zeile gesucht~~.

Default: 01

toffset **Gültige Werte:** 01 - 80

Mit *toffset* kann die Position bestimmt werden, wohin ein Feld hingestellt werden soll.

Default: 01

mark **Gültige Werte:** Maximal 7-stellige Sprungmarke (MARK / MAP)

mark definiert eine Sprungmarke auf die nach einer FOUND- oder NFOUND-Bedingung verzweigt wird, je nachdem, ob der definierte Wert *data* gefunden wurde oder nicht.

Beschreibung:

SCAN durchsucht den erhaltenen Bildschirminhalt (SCREEN) oder ein definiertes Feld (WORK1 - 9 oder DCL-Feld) auf einen bestimmten Wert. Mittels den FOUND- oder NFOUND-Parametern kann entsprechend auf eine Sprungmarke zur weiteren Verarbeitung verzweigt werden. Im Falle einer FOUND-Bedingung steht der Wert linksbündig im Bereich, der mit dem TO Parameter definiert wurde (Default: WORK1) und kann z.B. mit den Befehlen MOVE, PUTPRT etc. weiterverarbeitet werden.

Beispiele:

```
     :
     SCAN DATA='SIGN-ON IS COMPLETE',FOUND=OK1
     PUTPRT DATA='SIGN-ON FEHLER'
     GOTO MARK=ENDE
     :
OK1  MARK
     :
```

Abbildung 71. Beispiel 1 SCAN

```
       :
       MOVE FROM=SCREEN,LIN=04,COL=67,TO=S1
S1     SCAN DATA='99.999.999',FROM=SCREEN,      C
           LIN=05,COL=67,FOUND=OK1
       PUTPRT DATA='NICHT GEFUNDEN'
       :
OK1    MARK
       :
```

Abbildung 72. Beispiel 2 SCAN

```
       :
SWITCH DCL  DATA='ON'
       :
       PERFORM PROC=P1
       SCAN DATA='ON',FROM=SWITCH,FOUND=A1
       :
A1     MAP
       MAPEND
       :
P1     PROC
       :
       MOVE TO=SWITCH,DATA='OFF'
       :
       PROCEND
       :
```

Abbildung 73. Beispiel 3 SCAN

6.33. SCANB

Syntax:

	Befehl	*Parameter*
[*label*]	**SCANB**	**DATA**='*data*' [,**FROM**=*from* ,**TO**=*to* ,**LIN**=*line* ,**COL**=*col* ,**ENDLIN**=*endlin* ,**ENDCOL**=*endcol* ,**TOFFSET**=*toffset* ,**FOUND**=*mark* ,**NFOUND**=*mark*]

label **Gültige Werte:** Maximal 7 alphanumerische Zeichen.

label kann mit TO / FROM referenziert werden (z.B. mit MOVE).

data **Gültige Werte:** Maximal 80 Zeichen in Grossbuchstaben.

Der definierte Wert *data* wird auf dem empfangenen Bildschirminhalt gesucht.

Default: keiner

from **Gültige Werte:** SCREEN, WORK1 - 9 oder vordefiniertes Feld

from bezeichnet den Bereich, in dem der String im DATA Parameter gesucht wird.

Default: SCREEN (Bildschirm)

to **Gültige Werte:** WORK1 - 9 oder vordefiniertes Feld

to bezeichnet einen Bereich, in den die nächsten 80 Zeichen ab dem gefundenen String im DATA Parameter gestellt werden oder die Daten, die innerhalb der mit ENDLIN und ENDCOL definierten Position liegen. Bei WORK1 - 9 werden 80 Zeichen übertragen.

Default: WORK1

lin **Gültige Werte:** 01 - 43

lin definiert die Zeilenposition bei einem SCANB auf den SCREEN.

Default: 01

col **Gültige Werte:** 01 - 132

col definiert die Kolonnenposition, ab welcher der SCANB auf SCREEN oder WORK1 - 9 beginnen soll.

Default: 01

endlin **Gültige Werte:** 01 - 43

Mit *endlin* kann eine Endposition einer Zeile für den SCANB auf SCREEN definiert werden. Der String im DATA Parameter muss bei Angabe von ENDLIN innerhalb der Zeilen liegen, die mit LIN und ENDLIN definiert wurden. Trifft dies nicht zu, wird die NFOUND-Bedingung gesetzt. Der Wert *endlin* muss **grösser** oder **gleich** dem Wert in LIN sein.

Default: 01

endcol **Gültige Werte:** 01 - 132

Mit *endcol* kann eine Endposition einer Kolonne für den SCANB auf SCREEN oder WORK1 - 9 definiert werden. Der String im DATA Parameter muss bei Angabe von ENDCOL innerhalb einer Kolonne liegen, die zwischen COL und ENDCOL liegt. Trifft dies nicht zu, wird die NFOUND-Bedingung gesetzt. Der Wert muss **grösser** oder **gleich** dem Wert in COL sein. Falls *col* und *endcol* den gleichen Wert haben, wird der String auf der ganzen Zeile gesucht.

Default: 01

toffset **Gültige Werte:** 01 - 80

Mit *toffset* kann die Position bestimmt werden, wohin ein Feld hingestellt werden soll.

Default: 01

mark **Gültige Werte:** Maximal 7-stellige Sprungmarke (MARK / MAP)

mark definiert eine Sprungmarke auf die nach einer FOUND- oder NFOUND-Bedingung verzweigt wird, je nachdem, ob der definierte Wert *data* gefunden wurde oder nicht.

Beschreibung:

SCANB durchsucht den erhaltenen Bildschirminhalt oder ein definiertes Feld (WORK1 - 9 oder DCL-Feld) **rückwärts** (backwards) auf einen bestimmten Wert (Siehe SCAN auf Seite 102).

Beispiele analog SCAN ab Seite 101.

SAP: SCANB kann in SBDC-Transaktionen verwendet werden, um die letzte Mappe auf einer Seite zu positionieren.

6.34. SESSBEG

Syntax:

	Befehl	*Parameter*
	SESSBEG	[**SESSID=***sessid*]

sessid **Gültige Werte:** Maximal 1-stelliger Character

sessid zeigt, für welche Session SESSBEG gilt.

Default: A.

Beschreibung:

Der SESSBEG Befehl bestimmt den Beginn einer ATO Session. VORHER werdem mit SESSSET die Parameter für die Session festgelegt.

Beispiel:

```
SESSSET SESSID=A,                                                    *
      APPLID=DBDCCICS
SESSBEG
```

Abbildung 74. Beispiel SESSBEG

6.35. SESSEND

Syntax:

	Befehl	*Parameter*
	SESSEND	**SESSID=***sessid*

sessid **Gültige Werte:** Maximal 1-stelliger Character

Mit diesem Parameter wird angegeben, welche Session beendet werden soll.

Default: Gleich wie *sessid* aus SESSBEG.

Beschreibung:

Der SESSEND Befehl bestimmt das Ende einer einzelnen Session. Vor dem EPILOG Befehl müssen sämtliche ATO Sessions durch SESSEND beendet werden.

Beispiel:

```
SESSEND SESSID=A
```

Abbildung 75. Beispiel SESSEND

6.36. SESSSET / SESSMOD

Syntax:

	Befehl	*Parameter*
	SESSSET **SESSMOD**	**DIALOG=***name* ,**NETNAME=***netname* ,**APPLID=***applid* ,**APPLTRY=***appltry* ,**LOGTERM=***logterm* ,**LOGTIME=***logtime* ,**LOGTRY=***logtry* ,**LOOP=***loop* ,**LINEOV=***lineov* ,**MDTAUTO=***mdtauto* ,**TIMEOUT=***timeout* ,**SESSID=***sessid* ,**SUPPORT=***support* ,**LOGMODE=***logmode* ,**LOGTMOD=***logtmod*

name **Gültige Werte:** Maximal 8 alphanumerische Zeichen

Dieser Parameter bezeichnet den Namen des auszuführenden Dialoges.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

netname **Gültige Werte:** Maximal 8-stelliger Terminalname gemäss Netzwerk-Konventionen

CICS: Dieser Parameter bezeichnet einen gültigen im CICS definierten oder im AUTOINSTALL zugelassenen Netzwerknamen.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

applid **Gültige Werte:** APPLID Name gemäss Netzwerk-Konventionen

APPLID für eine ATO Session.

CICS: Dieser Parameter bezeichnet den in der SIT des entsprechenden CICS eingetragenen Applikationsnamen**.** Dieser ist auch mit der Transaktion CEMT INQ TAS am unteren rechten Bildrand ersichtlich.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

appltry **Gültige Werte:** 1 - 6000

appltry ist die Anzahl der Versuche, die ATO für die Herstellung einer Verbindung zur Applikation in Parameter APPLID durchführt.

Die Angabe von **APPLTRY=1** kann verwendet werden, um einen Dialog nur dann auszuführen, wenn der TP-Monitor aktiv und verfügbar ist.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

logterm **Gültige Werte:** Maximal 8 alphanumerische Zeichen. Gültiger Netzname gemäss Netzwerk-Konventionen, NO oder LOGON.

Dieser Parameter definiert den Netzwerknamen eines Bildschirms für die Überwachung und Schritt-für-Schritt-Anzeige dieses Dialoges. Ist diese Anzeige nicht erwünscht, so ist LOGTERM=NO zu spezifizieren.

Bei Angabe von LOGTERM=LOGON kann man sich dynamisch in den LOGTERM einschalten durch Eingabe von LOGON APPLID(*netname*). ATO wartet dabei mit der Verarbeitung , bis ein LOGON APPLID(*netname*) erfolgt.

Bemerkung:

LOGTERM ist auch für Session-Manager Produkte unterstützt.

Die Verwendung von LOGON APPLID(...) entspricht dem Default USSCMD-FORMAT=PL1 in der USSTAB Definition. Bei Angabe von FORMAT=BAL ist LOGON APPLID=... zu verwenden. Die Eingabe erfolgt bei USSTAB Message 10.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

logtime **Gültige Werte:** 0 - 60

logtime definiert eine Zeitverzögerung in Sekunden mit LOGTERM. Während dieser Zeit wartet ATO mit der Ausgabe des nächsten Bildes.
Mit dem Befehl LOGTIME kann der Wert im Dialog neu gesetzt und übersteuert werden (Siehe Befehl LOGTIME auf Seite 63).

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

logtry **Gültige Werte:** 0 - 6000

logtry definiert die Anzahl der Versuche, um eine Verbindung zum LOGTERM Bildschirm herzustellen. Bei Angabe von **LOGTRY=1** wird der LOGTERM Bildschirm nur verwendet, wenn dieser auch verfügbar ist bzw. dieser im VTAM definiert ist. Ansonsten wird die Verarbeitung ohne LOGTERM fortgesetzt.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

loop **Gültige Werte:** 1 - 100000

loop definiert den Maximalwert, den der Loop-Counter erreichen kann. Beim Erreichen dieses Werts wird der Dialog abgebrochen.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

lineov **Gültige Werte:** 1 - 99999

Mit *lineov* wird die Anzahl Zeilen pro Seite bestimmt.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

mdtauto **Gültige Werte:** YES, NO

Bei Angabe von MDTAUTO=YES werden für MDT-Bildschirmfelder automatisch FILL-Befehle generiert (z.B. CICS BMS-FSET Felder). Damit können auch Felder übertragen werden, die für den Benutzer auf dem Bildschirm unsichtbar (Dark) bleiben und nur der Dialogsteuerung dienen.

Bei Definition von MDTAUTO=NO werden nur Felder übertragen, die mit MAPFILL eingefügt oder verändert wurden.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

timeout **Gültige Werte:** 0 - 60000

Dieser Parameter definiert die maximale Zeit in Sekunden, während der ATO auf eine Antwort wartet. Dieser Parameter setzt den Default für MAP- und PENDING-TIMEOUT. Der Default-Wert genügt für die meisten Anwendungen.

Mit der Angabe von TIMEOUT=1440 wird die Zeitüberschreitungs-Kontrolle ausgeschaltet. TIMEOUT=1440 sollte nur für langlaufende Dialoge verwendet werden.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

sessid **Gültige Werte:** 1-stelliger Character

Dieser Parameter gibt die Session an, deren Parameter verändert werden sollen..

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

support **Gültige Werte:** NO, YES

Dieser Parameter dient der Diagnose-Hilfe. Bei Verwendung von SUPPORT=YES werden detaillierte Dialogauswertungen auf den ATOLOG geschrieben. Dabei werden auch die Werte aus dem User Exit ATOEXI ausgegeben.

Default: NO

VSE: Bei SUPPORT=YES muss der JCL-Parameter //OPTION LOG,PARTDUMP definiert sein.

MVS: Bei SUPPORT=YES muss der JCL-Parameter MSGLEVEL=(1,1), sowie //SYSUDUMP definiert sein.

logmode **Gültige Werte:** LOGMODE Name gemäss VTAM Definition

Mit diesem Parameter wird spezifiziert, welcher LOGMODE für den virtuellen Bildschrim gelten soll.

Es können alle Bildschirm-Modelle (Modell 2, 3, 4 oder 5) verwendet werden.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

logtmod **Gültige Werte:** LOGMODE Name gemäss VTAM Definition

Mit diesem Parameter wird spezifiziert, welcher LOGMODE für den LOGTERM-Bildschrim gelten soll. In der Regel sollte derselbe LOGMODE wie beim *logmode* Parameter angegeben werden.

Es können alle Bildschirm-Modelle (Modell 2, 3, 4 oder 5) verwendet werden.

Default: Wert aus PROLOG bzw. vorherigem SESSSET-Befehl

Beschreibung:

Der SESSET Befehl folgt dem SESSBEG und setzt alle Parameter fest, die von den im PROLOG festgelegten Werten abweichen. Während der Session können die Parameter mittels SESSMOD verändert werden.

Bemerkung:

Beispiel:

```
        SESSBEG SESSID=A
        SESSSET SESSID=A,                    C
             NETNAME=NETATO1,                C
             APPLID=DBDCCICS,                C
             LOGTERM=NO
        :
        SESSEND SESSID=A
        EPILOG
        END
```

Abbildung 76. Beispiel SESSSET

Funktion bei Aufruf aus einer Programmiersprache

ATO_SET_SESSION_PARAMETERS

6.37. SLEEP

Syntax:

	Befehl	*Parameter*
	SLEEP	**SEC**=*sec*

sec **Gültige Werte:** 1 - 3600.

Zeitwert in Sekunden für den SLEEP Befehl.

Default: 60

Beschreibung:

SLEEP dient zur Unterbrechung des laufenden Dialoges. Durch einen SLEEP-Befehl wird kein neuer Bildschirm eingelesen, daher ist nach einem SLEEP-Befehl erneut eine MAP -, MAPEND-Sequenz zu durchlaufen.

Beispiel:

```
        :
L1      MARK
        MAP  DATA='MENU'
        MAPEND
        SCAN DATA='HAUPT-MENU',FOUND=L2
        PUTPRT DATA='WARTEN BIS MENU ENABLED'
        SLEEP
        GOTO MARK=L1
        :
L2      MARK
        :
```

Abbildung 77. Beispiel SLEEP

6.38. UEXIT

Syntax:

Befehl	*Parameter*
UEXIT	**DATA**='*data*' [,**TO**=*to*]

data **Gültige Werte:** Maximal 80 alphanumerische Zeichen

Die Daten werden dem User Exit ATOEXI zur Steuerung übergeben, der den entsprechenden Wert in der mit *to* bezeichneten User Variablen zurückgeibt.

Default: Keiner

to **Gültige Werte:** WORK1 - 9, FILL-Label oder mit DCL deklariertes Feld.

to bezeichnet den Bereich, in den der User Exit ATOEXI die angeforderten Daten stellt.

Default: WORK1

Beschreibung:

Mit UEXIT wird der User-Exit ATOEXI explizit aufgerufen. ATOEXI dient zur Feldmanipulation mit diversen Möglichkeiten. Weitere Informationen betreffend ATOEXI sind im Anhang D auf Seite 129 zu finden.

Beispiel:

```
        PROLOG ...,SUPPORT=YES
        :
        UEXIT DATA='USER1',TO=WORK2
        PUTPRT FROM=WORK2
        MAP  DATA='MENU'
        MAPEND
        MOVE FROM=WORK2,TO=F1
        MAP
F1      FILL LIN=03,COL=04,DATA='XXXX'
        MAPEND
        :
```

Abbildung 78. Beispiel UEXIT

7. AUSFÜHREN VON ATO DIALOGEN

Dieses Kapitel behandelt die Vorgehensweise zur Ausführung von ***masc-ato*** Dialogen unter VSE oder MVS.

Die Einbettung in REXX kann in beliebige REXX Programme erfolgen. Jobs für den Aufruf stehen auf dern SAMPLIB zur Verfügung. Es ist keine Umwandlung oder Generierung notwendig.

Die ***masc-ato*** Befehle des Kompatibilitätsmodus sind als Assembler-Macros definiert und entsprechen den Assembler-Konventionen. Damit ein Dialog ausgeführt werden kann, müssen diese Macros (oder ***masc-ato*** Befehle) zuerst umgewandelt, d.h. assembliert und danach als Loadmodul gelinkt werden. Dabei ist zu beachten, dass bei Verletzung von Assembler-Konventionen (siehe Kapitel "Allgemeine Notations-Regeln" auf Seite 1) die entsprechenden Fehlermeldungen vom Assembler ausgegeben werden.

Grundsätzlich gibt es 2 Möglichkeiten, einen ***masc-ato***Dialog auszuführen:

- **Aufruf eines Dialoges im Link and Run-Mode**
- **Aufruf eines Dialoges im Go-Mode**

Generell ist die erste Methode zu bevorzugen. Eine Gegenüberstellung von Vor- und Nachteilen der Methoden finden Sie später in diesem Kapitel.

7.1. VSE Assembly mit Link and Run

Die in Abbildung 79 dargestellte JCL führt einen ATO Dialog aus, wobei die ATO Befehle zuerst assembliert, als Loadmodul gelinkt und danach ausgeführt werden.

```
* $$ JOB JNM=ATORUN
* $$ LST CLASS=L
* $$ LST CLASS=L,LST=01E
// JOB ATORUN
// OPTION LOG,PARTDUMP
// LIBDEF *,SEARCH=(userlib.ATO410),CATALOG=userlib.ATO410
// OPTION CATAL,NOXREF
   ACTION CLEAR
   PHASE DIALOG1,*
// EXEC ASSEMBLY
        PROLOG DIALOG=DIALOG1,...
        :
        EPILOG
        END
/*
// EXEC LNKEDT
// ASSIGN SYS001,01E
// ATORUN EXEC DIALOG1
040           * DISPONENT         (INPUT FUER GETRDR)
02401         * MATERIAL-NUMMER
/*
/&
* $$ EOJ
```

Abbildung 79. VSE Assembly mit Link and Go

7.2. MVS Assembly mit Link and Run

Die in Abbildung 80 dargestellte JCL führt einen ATO Dialog aus, wobei die ATO Befehle zuerst assembliert, als temporäres Loadmodul gelinkt und danach ausgeführt werden.

```
//ATORUN  JOB (ACCNT),'ATORUN',CLASS=A,MSGCLASS=X,
//          MSGLEVEL=(1,1),TIME=30
//PROC1   EXEC PROC=ATORUN,DIALOG=DIALOG1
//GEN.ATOCTL DD *
      PROLOG DIALOG=DIALOG1,...
      :
      EPILOG
      END
/*
//RUN.ATORDR DD *
040           * DISPONENT        (INPUT FUER GETRDR)
02401         * MATERIAL-NUMMER
/*
```

Abbildung 80. MVS Assembly mit Link and Go

7.3. MVS Dialog-Generierung mit getrennter Ausführung

Diese Methode setzt voraus, dass Sie die Source des ATO Dialoges assembliert und in eine User-Loadlibrary gelinkt haben. Das PARM-Feld in der EXEC Anweisung definiert den auszuführenden ATO Dialog mit DIALOG=***dialogname***, wobei ***dialogname*** dem Loadmodul-Name in der entsprechenden ***user.loadlib*** entspricht.

```
//ATOGEN  JOB (ACCNT),'ATOGEN',CLASS=A,MSGCLASS=X,
//          MSGLEVEL=(1,1)
//*
//ATOGEN  EXEC PROC=ATOGEN,DIALOG=DIALOG1,
//            LOAD=ato.user.loadlib
/*
```

Abbildung 81. MVS ATO Dialog-Generierung

```
//job      JOB  ...
//ATOGO      EXEC PROC=ATOGO,DIALOG=dialogname
//GO.STEPLIB DD DSN=user.loadlib,DISP=SHR
//GO.ATORDR DD *
040           * DISPONENT
02401         * MATERIAL-NUMMER
/*
```

Abbildung 82. MVS Loadmodul Ausführung

7.4. Vor- und Nachteile der ATO Dialog Ausführungsmethoden

Wie in den vorangehenden Kapiteln angesprochen, haben die 2 Ausführungsmethoden

- **Assembly mit Link and Run-Mode**

  und

- **Aufruf eines Dialogs im Go-Mode**

verschiedene Vor- und Nachteile. Wie bereits erwähnt, ist die erste Methode mit **Assembly und anschliessendem Link and Run** vorzuziehen, weil damit die Konsistenz aller Dialoge nach einem ATO Upgrade in eine neue Umgebung in jedem Fall gewährleistet ist und ein vorgängiges Umwandeln und Linken aller Dialoge entfällt.
Bei dieser Methode ist zudem der ganze Dialog in einem einzigen Job zusammengefasst, was bei Rückfragen und Änderungen hilfreich sein kann.

Die zweite Methode, der Aufruf eines bereits **assemblierten und gelinkten Loadmodules**, hat den Vorteil, dass Source- und Loadmodul getrennt sind, was bei Installationen mit strikter Gewaltentrennung gewünscht oder sogar verlangt wird. Ferner wird bei der Ausführung kein Assembly und Link-Step durchlaufen, was einer geringeren CPU-Belastung gleichkommt. Diese Ersparnis ist im MVS jedoch minimal.
Bei einem Release Wechsel von ATO sind die Dialoge neu zu assemblieren und zu linken.

8. ANHANG A

8.1. CICS An- und Abmeldung

8.1.1. Aufruf aus REXX

```
/* ******************************************************/
/* SAMPLE 1: SIGN-ON / SIGN-OFF TO CICS                 */
/* ******************************************************/
ARG ARG1 ARG2

CALL INITIALIZE
CALL PROLOG
CALL LOGON

CALL SIGNOFF
SAY 'NORMAL TERMINATION OF DIALOG, MAX_RC =' MAX_RC
EXIT MAX_RC

RETTEST:
/* CHECK THE ATO RETURN CODE */
  IF C2D(ATO_RC) > 0 THEN DO
     SAY 'ATO ERROR =' C2D(ATO_RC) ', FUNCTION' C2D(ATO_FUNC)
     RC = C2D(ATO_RC)
     IF RC > MAX_RC THEN MAX_RC = RC
  END
  RETURN

INITIALIZE:
/* LIST OF ALL FUNCTIONS AVAILABLE FOR ATO REXX INTERFACE */
  ATO_INITIALIZE             = D2C(1000,4)
  ATO_TERMINATE              = D2C(1001,4)
  ATO_CONNECT_PS             = D2C(01,4)
  ATO_DISCONNECT_PS          = D2C(02,4)
  ATO_SEND_KEY               = D2C(03,4)
  ATO_COPY_PS                = D2C(05,4)
  ATO_QUERY_CURSOR           = D2C(07,4)
  ATO_COPY_PS_TO_STRING      = D2C(08,4)
  ATO_SET_SESSION_PARAMETERS = D2C(09,4)
  ATO_COPY_STRING_TO_PS      = D2C(15,4)
  ATO_PAUSE                  = D2C(18,4)
  ATO_QUERY_SESSION_STATUS   = D2C(22,4)
  ATO_SET_CURSOR             = D2C(40,4)
  ATO_SEND_AID               = D2C(59,4)
  ATO_PENDING                = D2C(60,4)
  ATO_PUTLOG                 = D2C(62,4)
  ATO_PUTWTO                 = D2C(64,4)
  ATO_SLEEP                  = D2C(66,4)
  ATO_DISCONNECT_FORCE       = D2C(68,4)
  ATO_EXI                    = D2C(70,4)
  MAX_RC                     = 0
  RETURN

PROLOG:
  /* MAIN INITIALIZATION */
  /* THIS FUNCTION MUST BE EXECUTED FIRST IN ALL CASES */
  ATO_FUNC = ATO_INITIALIZE
  ATO_BUFF = 'TRACE=NO'
  CALL 'ATO'
  /* IN CASE OF ERROR WE MUST QUIT */
  IF C2D(ATO_RC) > 0 THEN DO
     SAY 'ATO ERROR NACH ATO_INITIALIZE =' C2D(ATO_RC)
     ATO_FUNC = ATO_TERMINATE
     ATO_BUFF = ''
     CALL 'ATO'
     EXIT 8
```

```
  END

  /* SET SESSION PARAMETERS FOR SESSION A */
  ATO_SID  = 'A'
  ATO_FUNC = ATO_SET_SESSION_PARAMETERS
  ATO_BUFF =          'NETNAME=A01ATO1,'
  ATO_BUFF = ATO_BUFF 'APPLID=DBDCCICS,'
  ATO_BUFF = ATO_BUFF 'LOGMODE=D4A32782,'
  ATO_BUFF = ATO_BUFF 'TIMEOUT=20,'
  ATO_BUFF = ATO_BUFF 'APPLTRY=4,'
  ATO_BUFF = ATO_BUFF 'SUPPORT=NO,'
  ATO_BUFF = ATO_BUFF 'LOGTERM=LOGON,'
  ATO_BUFF = ATO_BUFF 'LOGTIME=1,'
  ATO_BUFF = ATO_BUFF 'LOGTERMMODE=D4B32782,'
  ATO_BUFF = ATO_BUFF 'LOGTRY=4'
  CALL 'ATO'
  IF C2D(ATO_RC) > 0 THEN DO
     SAY 'ATO ERROR NACH ATO_SET_SESSION_PARAMETERS =' C2D(ATO_RC)
     EXIT
  END

  /* HERE WE CONNECT SESSION A */
  ATO_FUNC = ATO_CONNECT_PS
  CALL 'ATO'
  IF C2D(ATO_RC) > 0 THEN DO
     SAY 'ATO ERROR NACH ATO_CONNECT_PS =' C2D(ATO_RC)
     EXIT 8
  END
  RETURN

LOGON:
  /* CLEAR SCREEN */
  CALL AID CLEAR
  CALL RETTEST

  /* START CESN TRANSACTION */
  ATO_FUNC = ATO_SEND_KEY
  ATO_BUFF = 'CESN '
  CALL 'ATO'
  /* HIT ENTER */
  CALL AID ENTER

  /* TYPE USERID AND PASSWORD AT HOME AND HOME & NEWLINE RESP. */
  ATO_FUNC = ATO_SEND_KEY
  ATO_BUFF = '@HATOUSR1@H@NATO'
  CALL 'ATO'
  CALL AID ENTER

  /* CHECK, IF SIGN-ON OK */
  ATO_FUNC = ATO_QUERY_CURSOR
  CALL 'ATO'
  ATO_FUNC = ATO_COPY_PS_TO_STRING
  ATO_BUFF = COPIES(' ',19)
  NEW_POSITION = C2D(ATO_POSITION) - 61
  ATO_POSITION = D2C(NEW_POSITION,4)
  CALL 'ATO'
  IF ATO_BUFF ^= 'SIGN-ON IS COMPLETE' THEN
     DO
       SAY 'LOGON TO CICS FAILED'
     END
  CALL AID CLEAR
  RETURN

SIGNOFF:
  /* SIGN-OFF FROM CICS */
  CALL AID CLEAR
  ATO_FUNC = ATO_SEND_KEY
  ATO_BUFF = 'CESF LOGOFF'
  CALL 'ATO'
  CALL AID ENTER

  /* HERE WE DISCONNECT SESSION A */
  ATO_FUNC = ATO_DISCONNECT_PS
  CALL 'ATO'
  CALL RETTEST

  /* MAIN TERMINATION */
  /* THIS FUNCTION MUST BE EXECUTED LAST IN ALL CASES */
  ATO_FUNC = ATO_TERMINATE
  CALL 'ATO'
  CALL RETTEST
  RETURN
```

```
AID:
  /* SEND AID */
  ARG ATO_BUFF
  ATO_FUNC = ATO_SEND_AID
  CALL 'ATO'
  CALL RETTEST
  RETURN

DISPSCR:
  /* SUBROUTINE TO DISPLAY THE CURRENT SCREEN CONTENTS */
  SAY '*** SCREEN DISPLAY ***'
  ATO_FUNC = ATO_COPY_PS_TO_STRING
  ATO_BUFF = COPIES(' ', 4000)
  ATO_POSITION = D2C(1,4)
  CALL 'ATO'
  CALL RETTEST
  POS = 1
  DO WHILE POS < LENGTH(ATO_BUFF)
   SAY SUBSTR(ATO_BUFF,POS,80)
   POS = POS + 80
  END
  SAY '*** END OF SCREEN ***'
  RETURN
```

Abbildung 83. ATO Dialog Beispiel 1

Bemerkung

Dieses Beispiel eines ATO Dialogs führt ein SIGN-ON im DBDCCICS mit anschliessendem 'CESF LOGOFF' durch. Bei nicht erfolgreichem Sign-On wird ein Returncode 8 gesetzt.

Das Beispiel soll zeigen, wie Unterroutinen helfen, das Haupt-Programm von häufig vorkommenden Code-Sequenzen zu entlasten und damit übersichtlicher zu gestalten.

Die Angaben DATA='ATOUSR1' und DATA='ATO' können auch durch DATA='#VAR1#' bzw. DATA='#VAR2#' mittels dem ATO User-Exit ATOEXI übernommen werden. Vgl. dazu bitte den entsprechenden Anhang und das Kapitel ATO_EXI.

Der Dialog steht auf der ausgelieferten SAMPLIB mit dem Namen RXAPPA1.

8.1.2. Kompatibilitäts-Modus

```
        PROLOG DIALOG=SAMPLE1,                          C
              NETNAME=NETATO1,                          C
              APPLID=DBDCCICS
************************************************
* AUFRUF SIGN-ON TRANSAKTION                   *
************************************************
LOGON   MARK
MAP1    MAP  KEY=CLEAR
        MAPEND
        MAP  DATA='CESN'
        MAPEND
************************************************
* EINGABE USERID                               *
************************************************
MAP2    MAP
        FILL LIN=04,COL=14,DATA='ATOUSR1'
        FILL LIN=06,COL=16,DATA='ATO'
        MAPEND
************************************************
* SIGN-ON OK ? JA, SIGNOK                      *
************************************************
        SCAN DATA='SIGN-ON IS COMPLETE',FOUND=SIGNOK
        PUTPRT DATA='LOGON CESN TO CICS FAILED'
```

```
************************************************
* SIGN-ON FAILED, EXIT DIALOG                  *
************************************************
EXIT1   MAP KEY=PF3
        MAPEND
        MAP KEY=CLEAR
        MAPEND
EXIT2   MAP LASTMAP=YES,RC=8,DATA='CESF LOGOFF'
        MAPEND
************************************************
* SIGN-ON OK ? JA, SIGNOK                      *
************************************************
SIGNOK  MARK
LOGOFF  MARK
        MAP KEY=CLEAR
        MAPEND
MAP99   MAP LASTMAP=YES,DATA='CESF LOGOFF'
        MAPEND
        EPILOG
        END
```

Abbildung 84. ATO Dialog Beispiel 1

Bemerkung

Dieses Beispiel eines ATO Dialogs führt ein SIGN-ON im DBDCCICS mit anschliessendem 'CESF LOGOFF' durch. Bei nicht erfolgreichem Sign-On wird ein Returncode 8 gesetzt.
Die Angaben DATA='ATOUSR1' und DATA='ATO' können auch durch DATA='#VAR1#' bzw. DATA='#VAR2#' mittels dem ATO User-Exit ATOEXI übernommen werden.

8.2. CICS Befehl CEMT INQ TAS

8.2.1. Aufruf aus REXX

```
/* ******************************************************/
/* SAMPLE 2: CICS TRANSACTION 'CEMT I TA'                */
/* ******************************************************/
ARG ARG1 ARG2

CALL INITIALIZE
CALL PROLOG
CALL LOGON

/* START CEMT TRANSACTION */
ATO_FUNC = ATO_SEND_KEY
ATO_BUFF = 'CEMT I TA'
CALL 'ATO'
CALL AID ENTER
CALL DISPSCR
CALL PF3

CALL SIGNOFF
SAY 'NORMAL TERMINATION OF DIALOG, MAX_RC =' MAX_RC
EXIT MAX_RC

...
```

Abbildung 85. ATO Dialog Beispiel 2 aus REXX

Das Beispiel steht auf der ausgelieferten SAMPLIB unter dem Namen RXAPPA2.

8.2.2. Kompatiblitäts-Modus

```
       PROLOG DIALOG=SAMPLE2,                          C
            NETNAME=NETATO1                            C
            APPLID=DBDCCICS
************************************************
* ABFRAGE DER TASKS                             *
************************************************
LOGON   MARK
        :
        MAP  KEY=CLEAR
        MAPEND
MAP1    MAP  DATA='CEMT INQ TAS'
        MAPEND
        MAP KEY=PF3
        MAPEND
************************************************
LOGOFF  MARK
        :
        EPILOG
        END
```

Abbildung 86. ATO Dialog Beispiel 2

8.3. CICS Befehle allgemein

8.3.1. Aufruf aus REXX

```
/* ******************************************************/
/* SAMPLE 3: EXECUTE TRANSACTION READ FROM SYSTSIN       */
/* ******************************************************/
ARG ARG1 ARG2

CALL INITIALIZE
CALL PROLOG
CALL LOGON

/* READ ONE INPUT LINE */
PULL ATO_BUFF
/* TYPE INPUT LINE AS TRANSACTION
ATO_FUNC = ATO_SEND_KEY
CALL 'ATO'
CALL AID ENTER
CALL DISPSCR
CALL AID PF3

CALL SIGNOFF
SAY 'NORMAL TERMINATION OF DIALOG, MAX_RC =' MAX_RC
EXIT MAX_RC

...

JCL-SYSTSIN Daten:

CEMT SET DAT(FI*) CLOSE
/*
```

Abbildung 87. ATO Dialog Beispiel 3 aus REXX

Das Beispiel steht auf der ausgelieferten SAMPLIB unter dem Namen RXAPPA3.

8.3.2. Kompatibilitäts-Modus

```
       PROLOG DIALOG=SAMPLE3,                        C
            NETNAME=NETATO1,                         C
            APPLID=DBDCCICS
************************************************
* EINLESEN EINER STEUERKARTE INS FELD CICSCMD   *
************************************************
LOGON   MARK
        :
        GETRDR TO=CICSCMD,EOF=NODATA
************************************************
* AUSFUEHREN DES EINGELESENEN BEFEHLES         *
************************************************
        MAP  KEY=CLEAR
        MAPEND
CICSCMD MAP  DATA='CEMT XXXXXXXXXXX'
        MAPEND
************************************************
* VERLASSEN BILDSCHIRMANZEIGE MIT PF3 UND MELDUNG  *
************************************************
        MAP KEY=PF3
        MAPEND
```

```
        PUTPRT DATA='---- BEFEHL AUSGEFUEHRT ----'
        GOTO MARK=ENDE
************************************************
* KEINE EINGABEDATEN ERHALTEN                  *
************************************************
NODATA  MARK
        PUTPRT DATA='---- KEINE BEFEHLE ----'
        MAP KEY=CLEAR
        MAPEND
        MAP LASTMAP=YES,RC=4,DATA='CESF LOGOFF'
        MAPEND
************************************************
* ABSCHLUSS DES ATO DIALOGES                   *
************************************************
ENDE    MARK
************************************************
LOGOFF  MARK
        :
        EPILOG
        END
```

```
JCL-GETRDR-Daten:

CEMT SET DAT(FI*) CLOSE
/*
```

Abbildung 88. ATO Dialog Beispiel 3

9. ANHANG B

9.1. Migration auf Version 4.1

masc-ato 4.1.0 ist source-kompatibel zur Vorversion. Alle Dialoge, die umgestellt werden sollen, müssen neu generiert werden.

Beachten Sie bitte, dass der Dialogname neu direkt beim Aufruf angegeben wird, in der Vorversion wurde er als Parameter mitgegeben.

10. ANHANG C

10.1. User Exit ATOEXI

Der User Exit ATOEXI steht für die verschiedensten Anforderungen zur Verfügung. ATOEXI wird an verschiedenen Stellen eines ATO Dialoges implizit durch ATO oder explizit unter Verwendung des UEXIT Befehles oder von FILL #VARn#-Variablen aufgerufen. System intern wird der User Exit ATOEXI an verschiedenen Punkten des Session-Handlings mit #SYSn# aufgerufen. Die aktuelle APPLID aus dem TP-Monitor Bind wird beim Session Passing automatisch nachgeführt.

Nach jeder Änderung muss der User Exit ATOEXI neu generiert werden und zwar gemäss Abbildung 86 für VSE und Abbildung 87 für MVS. Ersetzen Sie die Library oder Dataset Namen entsprechend Ihrer ATO Installation.

```
* $$ JOB ATOEXI
// JOB ATOEXI
// OPTION LOG
// LIBDEF *,SEARCH=(userlib.ATO410),CATALOG=userlib.ATO410
// OPTION CATAL,XREF
   ACTION CLEAR
   PHASE ATOEXI,*
// EXEC ASSEMBLY
        COPY ATOEXI
        END
/*
// EXEC LNKEDT
/*
/&
* $$ EOJ
```

Abbildung 89. ATOEXI Generierung im VSE

```
//ATOEXI  JOB  (ACCT#),'ATO EXIT',CLASS=A,MSGCLASS=X,
//             MSGLEVEL=(1,1)
//*******************************************************
//* ASSEMBLE AND LINK ATOEXI                            *
//*******************************************************
//ASMATO  EXEC PGM=IEV90,
//        PARM='DECK,NOLOAD,NOXREF,NOESD,NORLD'
//SYSIN     DD DSN=prefix.orig410.asm(ATOEXI),DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSLIB    DD DSN=prefix.orig410.asm,DISP=SHR
//          DD DSN=prefix.orig410.maclib,DISP=SHR
//          DD DSN=SYS1.MACLIB,DISP=SHR
//*         DD DSN=SYS1.SISTMAC1,DISP=SHR     * ATOM MACROS
//          DD DSN=SYS1.MODGEN,DISP=SHR
//SYSUT1    DD UNIT=VIO,SPACE=(CYL,(5,5))
//SYSUT2    DD UNIT=VIO,SPACE=(CYL,(5,5))
//SYSUT3    DD UNIT=VIO,SPACE=(CYL,(5,5))
//SYSPUNCH  DD DSN=&&ATO,DISP=(,PASS),UNIT=VIO,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
//             SPACE=(400,(50,50))
//LNKATO  EXEC PGM=IEWL,COND=(0,NE)
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD UNIT=VIO,SPACE=(1024,(20,20))
//SYSLIN    DD DSN=&&ATO,DISP=(OLD,DELETE)
//SYSLMOD   DD DSN=prefix.orig410.load(ATOEXI),DISP=SHR
```

Abbildung 90. ATOEXI Generierung im MVS

10.2. User Exit Variablen #VARn#

Die User Variablen #VAR1# - #VAR9# können aus REXX mit dem Aufruf von ATO_EXI gefüllt oder im Kompatibilitäts-Modus im FILL DATA Parameter verwendet werden. Damit werden vorgegebene Werte im ATOEXI abhängig vom Dialog-Namen, APPLID, User-ID etc. gesetzt. Die Abhängigkeit und die Übergabe wird im User Exit ATOEXI mit einer Tabelle bestimmt.

Die Übergabe von Variablen kann zur Eingabe von User-ID, Passwort, Accounting-Informationen, Mandant etc. werden. Die Abbildung 89 weiter unten zeigt einen Ausschnitt der Tabelle TAB des ATOEXI für diesen Zweck. Dieser Exit wird als Beispiel ATOEXI zusammen mit der ATO Software ausgeliefert. Er kann entsprechend Ihren Anforderungen auf Ihre Konfiguration angepasst und neu generiert werden.

Sowohl für den Kompatibilitäts-Modus wie auch für den Aufruf aus REXX wird dasselbe Modul von ATOEXI verwendet.

10.3. User Code im Exit

Der User Exit ATOEXI kann ebenfalls dazu verwendet werden, eigenen Code einzufügen.

Die dafür vorgesehene Stelle ist markiert mit dem Kommentar "`INSERT YOUR CODE HERE`" bzw. "`END USER CODE`".

Der Aufruf erfolgt aus REXX durch ATO_EXI mit dem Übergabebereich auf ATO_BUFF. Im Kompatibilitätsmodus wird der Exit durch `UEXIT DATA='...'` aufgerufen.

10.4. User Exit Anwendung

Anhand des folgenden Beispieles soll die Anwendung und Funktionsweise des User Exits ATOEXI im Kompatibilitäts-Modus erläutert werden. Für den Aufruf aus REXX vgl. die Funktion ATO_EXI.

```
         PROLOG DIALOG=ATODLG1,                                C
               NETNAME=NETATO1,                                C
               APPLID=CICSP1
LOGON    MAP   DATA='CESN'
         MAPEND
MAP10    MAP
         FILL LIN=04,COL=14,DATA='#VAR1#'   USERID
         FILL LIN=06,COL=16,DATA='#VAR2#'   PSW
         MAPEND
         :
```

Abbildung 91. User Exit Anwendung

Durch Definition der Variablen #VAR1# und #VAR2# in den DATA Parameter der FILL Befehle wird ATOEXI implizit aufgerufen. Zur Ausführungszeit wird im ATOEXI die Tabelle TAB gemäss Abbildung 88 durchsucht. Im obigen Beispiel werden die Variablen #VAR1# und #VAR2# durch die Werte MUSTER und ATO ersetzt. Generell können mit dieser Tabelle beliebige Variable zugeordnet werden, falls die entsprechenden Bedingungen erfüllt sind. Die entsprechenden Ersatzwerte sind in der Kolonne USER zu definieren. Pro Variable #VARn# (max. 9) und zu erfüllender Bedingung ist je eine Zeile in der Tabelle TAB im ATOEXI zu definieren. Die Länge der Zeile und der einzelnen Felder innerhalb einer Zeile sind exakt einzuhalten.

Die Definition einer Zeile ist wie folgt festgelegt:

```
0        1         2         3         4         5         6
1...+....0....+....0....+....0....+....0....+....0....+....0....+...
_____________________________________________________________________

*                DIALOG  NETNAME APPLID  VAR     USER DATA
         DC    C'DDDDDDDDNNNNNNNNAAAAAAAA#VARn#**UUUUUUUU'
_____________________________________________________________________

D      = Dialogname definiert in PROLOG DIALOG=
N      = Netname definiert in PROLOG NETNAME=
A      = Application ID definiert in PROLOG APPLID=
#VARn# = Variablen Nummer wobei für n eine Zahl von 1-9 steht
U      = Ersatzwert für #VARn#
```

Die Feldwerte können auch generisch mit '*' definiert werden, wobei die Asteriks (*) die nicht relevanten Positionen darstellen.

Die Tabelle TAB wird von oben nach unten (top-down) durchlaufen bis zum ersten, zutreffenden Eintrag. Dieser wird dem Dialog im FILL-DATA der entsprechenden #VARn#-Variablen zurückgegeben.

Falsch:

```
         DC    C'DIALOG1*NETATO1********#VARn#**UUUUUUUU'
         DC    C'DIALOG1*NETATO1*CICS1***#VARn#**UUUUUUUU'
```

Richtig:

```
         DC    C'DIALOG1*NETATO1*CICS1***#VARn#**UUUUUUUU'
         DC    C'DIALOG1*NETATO1********#VARn#**UUUUUUUU'

****************************************
* TABELLE VON ATO FILL VARIABLEN
****************************************
         PRINT OFF     DO NOW SHOW THIS VARIABLES
*                DIALOG  NETNAME APPLID  VAR     USER
         DS    0D
TAB      DS    0C
*                DIALOG  NETNAME APPLID  VAR     USER
         DC    C'********NETATO0*CICSP1**#VAR1#**MUSTER**'
         DC    C'********NETATO0*CICSP1**#VAR2#**ATO*****'
         DC    C'********NETATO0*CICSP1**#VAR3#**02******'
*                DIALOG  NETNAME APPLID  VAR     USER
         DC    C'********NETATO1********#VAR1#**ATO1****'
         DC    C'********NETATO1********#VAR2#**ATO*****'
         DC    C'********NETATO1********#VAR3#**02******'
*                DIALOG  NETNAME APPLID  VAR     USER
         DC    C'********NETATO2********#VAR1#**ATO2****'
         DC    C'********NETATO2********#VAR2#**ATO*****'
         DC    C'********NETATO2********#VAR3#**02******'
         :
         :
*                DIALOG  NETNAME APPLID  VAR     USER
         DC    C'********NETATO9********#VAR1#**ATO9****'
         DC    C'********NETATO9********#VAR2#**ATO*****'
         DC    C'********NETATO9********#VAR3#**02******'
*                DIALOG  NETNAME APPLID  VAR     USER
         DC    C'********NETATOA********#VAR1#**ATOA****'
         DC    C'********NETATOA********#VAR2#**ATO*****'
         DC    C'********NETATOA********#VAR3#**02******'
         :
         :
```

Abbildung 92. User Exit ATOEXI (Ausschnitt Tabelle)

11. INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

S

U

V

W