masc-ato

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

MATO-UG410-1-E

Distributor:	***masc ag*** Dept. SWD Birkenstr. 49 CH-6343 Rotkreuz (Switzerland)
Telephone:	041 / 790 53 44 International: (+41) 41 790 53 44
Telefax:	041 / 790 53 40 International: (+41) 41 790 53 40
Office hours:	8 - 12h, 14 - 17h CET (Mo - Fr)

March 1998 Edition.

Further copies of this documentation may be ordered with the enclosed order form.

PREFACE

This manual describes the Automated Transaction Operator (ATO) commands and how to use them. It contains further appendixes with :

- **Appendix A: Examples of ATO dialogs**
- **Appendix B: Migration hints for users of the previous version**
- **Appendix C: User Exit ATOEXI**

Changes to this publication are summarized under the heading "Summary of Changes". Technical changes are marked with a vertical bar | in the left-hand margin.

Readers of this publication should have fundamental knowledge of VSE- resp. MVS functions, as well as experience with the user interface of the corresponding application.

Knowledge of REXX or another programming language is also necessary.

Summary of ATO documentation

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

UPGRADE OVERVIEW

This is the first edition of "***masc-ato***: *Automated Transaction Operator Users Guide".*

Enhancements

masc-ato has been reworked extensively and has been supplemented with new commands and parameters. The new command options have been set up in a way that their correlation is logical in many places. Special attention has been paid to clarity and legibility of the commands.

We thank all users whose suggestions have assisted us in the writing of this new upgrade.

CICS: The examples in this documentation are based on CICS/ESA.

Changes in the March 1998 Edition

A new ATO_SET_SESSION_PARAMETERS parameter LOGAID shows the key pressed when following a session in logterm mode.

New PROLOG parameters:
- ☞ LOGAID shows the key pressedd when following a session in logterm mode
- ☞ MLOG shows all macros passeed during the session on ATOPRT
- ☞ ASMGEN and ASMDATA control the output of the assembly step

PUTPRT now accepts a label that can be used in GOTOs

The default values of LIN/ENDLIN and COL/ENDCOL change according to the values entered

TABLE OF CONTENTS

TABLE OF FIGURES

1. NOTATION-CONVENTIONS

The format of the ATO commands are supplemented by means of the following symbols:

- Square brackets [] specify optional fields or parameters.
- A vertical bar | divides up a selection. In the case that nothing else is mentioned, only one alternative may be chosen.
- **BOLD** letters must be entered exactly as described.
- *Italic* letters describe fields or parameters that have to be given together with a command.
- Application specific additions:

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

2. GENERAL NOTATION RULES

2.1. General Rules

The notation rules of ***masc-ato*** do not require a certain form but are in accordance to the rules of the corresponding programming language.

2.2. Compatibility Mode

The notation rules of the compatibility mode are in accordance to the assembler conventions. This means:

- Label start at column 1.
- Commands start at column 10.
- Parameters start at column 16.
- Continuations are characterized with a comma and a continuation character at column 72.

Only the commands documented in this manual should be used. This will ensure that the defined dialogs will be compatible with future developments.

```
        1     1   2         3         4         5         6         7 7
1...+....0....+6...0....+....0....+....0....+....0....+....0....+....0.2
_____________________________________________________________________________
Label    Cmd   Parameter1,                                                  C
               Parameter2,                                                  C
               Parameter3
```

Figure 1. General Notation Rules

Example:

```
        1     1   2         3         4         5         6         7 7
1...+....0....+6...0....+....0....+....0....+....0....+....0....+....0.2
_____________________________________________________________________________
MAP10    MAP
         FILL  DATA='LOGOFF',                                              C
               LIN=24,COL=05
         MAPEND
         MOVE  TO=FELD1,DATA='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXC
               XXXXXXXXX'
```

Figure 2. Example Notation Rules

Note:

- **o** All labels, commands and parameters should be entered in capital letters.
- **o** Unless otherwise noted the string in the DATA-parameter may contain a maximum of 80 characters and may be referenced with FROM/TO if a label has been defined.
- **o** A * in column 1 defines a comment line.
- **o** Special characters have to be avoided; the rules apply according to the Assembler conventions.

3. STRUCTURE

3.1. *masc-ato* commands

To control the sessions and screen interactions, ***masc-ato*** uses a set of clear functions that lets you write easily legible programs. All functions are described in detail in the following chapters.

In the "native" mode, a function is set and the buffer prepared with its parameters before the call to the module "ATO" is made. To call specific functions we recommend to use mnemotechnically easy names, that you may put into copybooks according to the rules of the programming language. The terminology used in this manual is well suited for the embedding in REXX programs.

3.2. Reserved Words

The following variables are reserved:

ATO_SID	Session ID
ATO_FUNC	Function
ATO_RC	Return-Code
ATO_POSITION	Cursor Position
ATO_BUFF	Parameters for a Function

3.3. Keys and cursor control

masc-ato can not only send "normal" keys to the screen, but also tabulator and cursor control keys. This allows to write dialogs in a user-friendly way.

To recognize these special keys, the "@" is used. This Escape-Character shows, that the next character has a special meaning.

The following entries are possible:

@h	**Home**	Set cursor to the first entry field of the screen
@n	**Newline**	Set cursor to the first entry field of the next line on the screen
@t	**Tabulator**	Set cursor to the next entry field
@r	**Cursor right**	Set cursor one character to the right
@l	**Cursor left**	Set cursor one character to the left
@u	**Cursor up**	Set cursor one line up
@d	**Cursor down**	Set cursor one line down

These special characters are behaving like the corresponding keys. Some keys may only skip to entry fields while other ones leave the cursor on protected areas.

The cursor is also moved like with manual entries. If for example an eight-character-field is filled with an eight-character-value, the cursor automatically skips to the next entry field.

The PF, PA, and Clear keys are sent with the ATO_SEND_AID command after filling the screen with any contents and setting the cursor to a certain place.

3.4. Program logic

masc-ato provides the interface to your on-line applications. The actual program logic is programmed in a programming language that is better suited for that purpose and also provides the necessary interfaces to system resources like databases, files, etc.

Additionally to loops, case structures and many possibilities to manipulate variables, REXX for example also provides a rich set of functions that can help enormously interpreting the received or preparing the next screen.

The commands described above are therefore integrated either in REXX, a relatively easy-to-use but powerful language portable to several platforms, or in another programming language like COBOL, Assembler, or PL/1.

The choice of the right programming language for your dialog depends on different factors. Portability, the rich set of functions, and the ease to develop dialogs may often be in favor of REXX while the access to DB/2 or VSAM datasets may be easier to program in other languages.

3.5. Sample *masc-ato* dialog

The following illustration shows a simple part of a dialog using ***masc-ato*** commands:

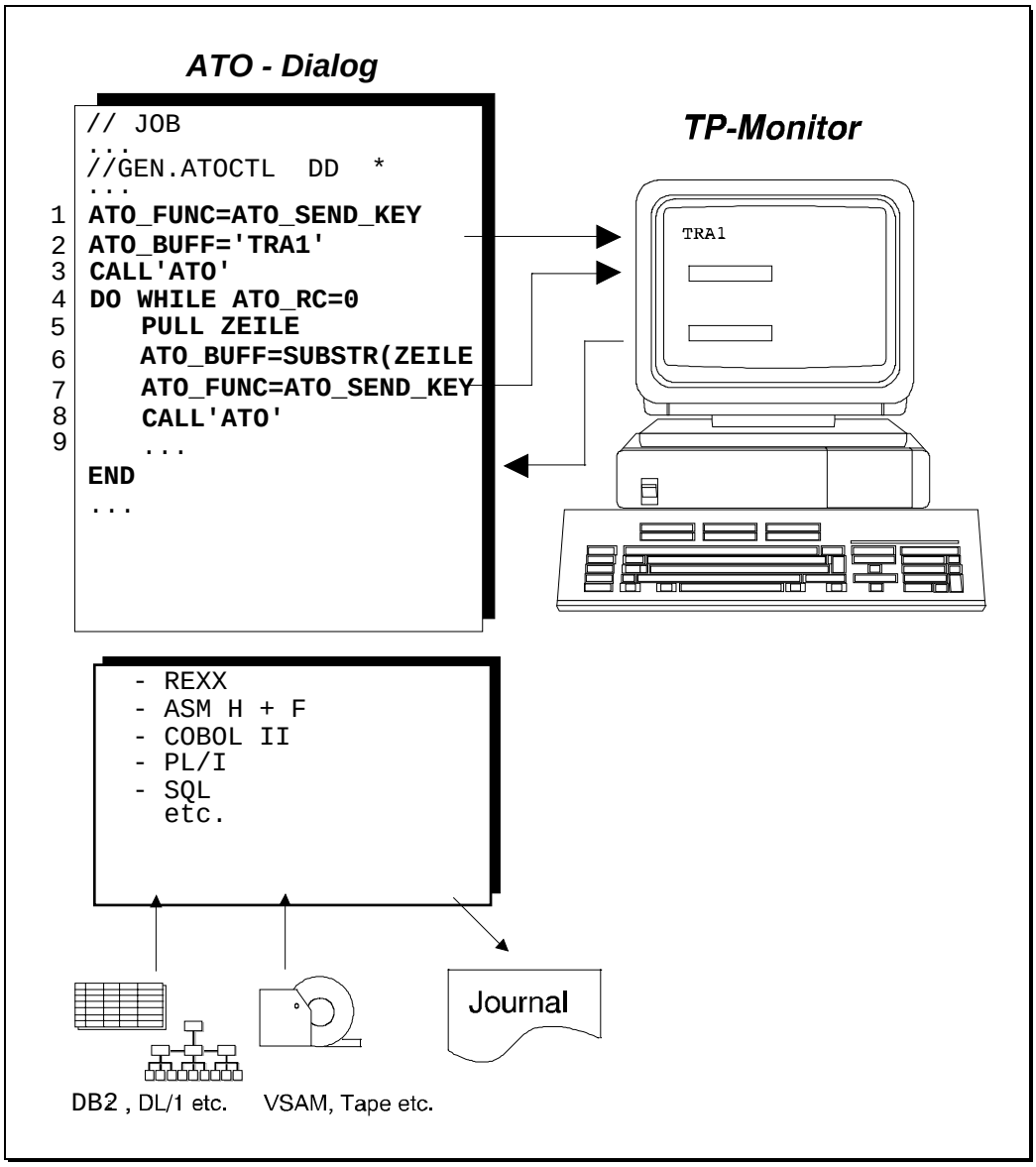


Figure 3. ***masc-ato*** Dialog

Explanations:

1 Prepare function *Send an entry to the virtual screen*
2 Put the desired entry into the buffer ...
3 ... and call ATO
4 Begin a loop including the check of the ATO return code
5 Read a line
6 Put part of this line into the buffer, ...
7 ... in order to have it appear as keyed-in entry on the virtual screen
8 Call ATO
9 ... many more ATO or REXX commands or functions

Each ***masc-ato*** dialog starts with a PROLOG- and ends with an EPILOG-command.

A ***masc-ato*** dialog consists of the session handling whereas for the other program logic a programming language like REXX is used.

Each session starts with a SESSBEG- and ends with a SESSEND-command. Up to 9 sessions can be used simultanesouly, the session-oriented commands are directed towards a specific session using the SESSID parameter.

Sequentially any number of sessions can be processed.

4. COMMAND OVERVIEW

This chapter gives an overview of the commands and functions of ***masc-ato***. Where useful the call from REXX and the macro in the compatibility mode are compared. The detailed description with all parameters follows in the chapters below.

4.1. Dialog and Session Control

Call from REXX	**Compatibility Mode**	
ATO_INITIALIZE	**PROLOG**	Initial values of a dialog
ATO_TERMINATE	**EPILOG**	End of a dialog
ATO_DISCONNECT_FORCE	**ABORT**	Cancel a dialog
(EXIT)	**END**	End
ATO_CONNECT_PS	**SESSBEG**	Begin of a specific session New in ***masc-ato*** 4.1.0
ATO_DISCONNECT_PS	**SESSEND**	End of a specific session New in ***masc-ato*** 4.1.0
ATO_QUERY_SESSION_STATUS	**n/a**	Query status of a session
ATO_SET_SESSION_PARAMETERS	**SESSSET**	Set session defaults New in ***masc-ato*** 4.1.0

Figure 4. Dialog and Session Control

4.2. Control of Virtual Screens

4.2.1. Call from REXX

ATO_QUERY_CURSOR		Query Cursor Position
ATO_SEND-KEY		Send keys to virtual screen
ATO_SET_CURSOR		Set Cursor Position
ATO_SEND_AID		Send Enter, PF- or PA-key to virtual screen
ATO_COPY_PS		Copy contents of virtual screen
ATO_COPY_PS_TO_STRING		Copy contents of virtual screen to buffer
ATO_COPY_STRING_TO_PS		Copy buffer to virtual screen

Figure 5. Dialog and Session Control

4.2.2. Compatibility Mode

MAP	Begin of screen input
MAPEND	End of screen input
FILL	Filling a screen display input field
MAPKEYS	Filling a screen display input field using keys and cursor control New in ***masc-ato*** 4.1.0
MAPFLD	Filling a screen input subfield

Figure 6. ATO Command Overview

4.3. Additional Commands

4.3.1. Call from REXX

ATO_PUTLOG	**PUTLOG**	Write Message to ATO Log
ATO_PUTWTO	**PUTWTO**	Write Message to Console
ATO_PENDING	**PENDING**	Process of pending transactions
ATO_PAUSE **ATO_SLEEP**	**SLEEP**	Sleep
ATO_EXI	**UEXIT**	Call User Exit ATOEXI

Figure 7. Additional Commands from REXX

4.3.2. Comaptibility Mode

	SCAN	Search in the output received
	SCANB	Search backwards in the output received
	GETRDR	Reading input data from ATORDR
ATO_PUTLOG	**PUTLOG**	Write Message to ATO LOG
	PUTPRT	Write Message to ATO PRT
ATO_PUTWTO	**PUTWTO**	Write Message to Console
	DCL	Declare variables
	GOTO	Branching to a branch mark
	LOOP	Reset the LOOP counter to a new value
	MARK	Branch mark
	PERFORM	Perform a procedure or subroutine
	PROC	Marks the begin of a procedure
	PROCEND	Marks the end of a procedure
	COPY	Copying of predefined dialog sequences (Copybook)
	HLLCALL	Communicating with a calling user program via HLL interface
	LOGTIME	Sets the dealy time between two commands when using LOGTERM

	MOVE	Transfer from and to WORK area
ATO_PENDING	**PENDING**	Processing of pending transactions
ATO_PAUSE **ATO_SLEEP**	**SLEEP**	Waiting time
ATO_EXI	**UEXIT**	Execution call User Exit ATOEXI
	ATOHLL	HLL Interface to call an ATO dialog from a user batch program with CALL

Figure 8. ATO Command Overview

5. COMMANDS

5.1. ATO_CONNECT_PS

Syntax:

ATO_FUNC	**ATO_CONNECT_PS**
ATO_SID	*sessid*

sessid **Valid values:** 1 character

sessid shows, for which session SESSBEG is valid.

Default: Like in ATO_SET_SESSION_PARAMETERS

Description:

The ATO_CONNECT_PS command defines the begin of a session. BEFORE, ATO_SET_SESSION_PARAMETERS sets all parameters for the session.

Example:

```
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
```

Figure 9. Example ATO_CONNECT_PS

5.2. ATO_COPY_PS

Syntax:

ATO_FUNC	**ATO_COPY_PS**
ATO_BUFF	*area*

area **Valid values:** Any area

area defines where to copy the virtual screen.

Description:

This function copies a virtual screen. In order to copy only parts of a virtual screen, ATO_COPY_PS_TO_STRING should be used instead.

Example:

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

Figure 10. Example ATO_COPY_PS

5.3. ATO_COPY_PS_TO_STRING

Syntax:

ATO_FUNC	**ATO_COPY_PS_TO_STRING**
ATO_BUFF	

Description:

This command copies the virtual screen content into a buffer where it can be used for further processing. The length of the buffer should be big enough to hold the entire screen. In order to avoid unpredictable results, the buffer should be initialized before calling the function.

A use of the ATO_COPY_PS_TO_STRING command is shown in figure 11. The example copies the virtual screen and afterwards displays it line by line. This routine can be used to show a hardcopy of the virtual screen.

Example:

```
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 ***'
```

Figure 11. Example 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	

Description:

This function serves to copy a buffer into a virtual screen. This allows to fill the virtual screen before sending it to the host using ATO_SEND_AID.

An example of the ATO_COPY_STRING_TO_PS command is shown in figure 12. This puts the string *nend* on line 24, column 5 of the virtual screen and sends it afterwards to the host pressing the ENTER key.

Example:

```
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'
```

Figure 12. Example ATO_COPY_STRING_TO_PS

5.5. ATO_DISCONNECT_FORCE

Syntax:

ATO_FUNC	**ATO_DISCONNECT_FORCE**
ATO_BUFF	

Description:

ATO_DISCONNECT_FORCE terminates the connection to the current session, however, the dialog or the program respectively continues.

A use of the ATO_DISCONNECT_FORCE command is shown in figure 13.

Example:

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

Figure 13. Example ATO_DISCONNECT_FORCE

Note:

ATO_DISCONNECT_FORCE does not sign-off or logoff from the TP-monitor, it is rather comparable to turning off the screen. Therefore, this command should be avoided.

5.6. ATO_DISCONNECT_PS

Syntax:

ATO_SID	*sessid*
ATO_FUNC	**ATO_DISCONNECT_PS**
ATO_BUFF	

sessid **Valid values:** 1 character

This parameter shows which session should be terminated.

Default: Like *sessid* in SESSBEG.

Description:

The ATO_DISCONNECT_PS command determines the end of a single session. Before ending the program, all sessions should be ended using ATO_DISCONNECT_PS.

Example:

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

Figure 14. Example ATO_DISCONNECT_PS

5.7. ATO_EXI

Syntax:

ATO_FUNC	**ATO_EXI**
ATO_BUFF	

Description:

The function ATO_EXI calls the user exit ATOEXI. The exit determines the communication area.

If ATO_BUFF is filled with one of the constants #VARx#, it will be filled with the value stored int the table of program ATOEXI after the call.

The following figure shows an example for the ATO_EXI command. #VAR2# is filled with the value from the ATOEXI table and entered as password on the screen.

Example:

```
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'
```

Figure 15. ExampleATO_EXI

5.8. ATO_INITIALIZE

Syntax:

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

trace **Valid values:** NO, YES

This parameter serves to swith on or off the ATOLOG. If *trace* is set to YES, all virtual screens of the dialog are shown.

Default: YES

Description:

The ATO_INITIALIZE command defines the begin of an ATO dialog and has to be the first ATO command that is processed. Besides the general initialization it also serves to swith on or off the trace.

Example:

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

Figure 16. Example ATO_INITIALIZE

5.9. ATO_PAUSE

Syntax:

ATO_FUNC	**ATO_PAUSE**
ATO_BUFF	

Description:

ATO_PAUSE interrupts the dialog until the next screen from the host is received. This allows to "wait" for certain events like asynchronous messages that a process has ended.

If *autopend* in ATO_SET_SESSION_PARAMETERS is not specified or is set to NO, the new virtual screen content is made available to the application program using ATO_PENDING. If *autopend* is set to YES, however, the virtual screen is immediately filled with the new content, there is no further action necessary.

The function ATO_PAUSE only ends when an asynchronous message arrives or if it gets a timeout. If the objective is to interrupt the dialog flow for a certain time, consider the use of ATO_SLEEP.

SAP: To wait for the end of a long running task, ATO_PAUSE can be used.

The following figure shows an example for ATO_PAUSE. After ATO_PAUSE ends because of the arrival of the next host message, it checks if it now received the next screen or an asynchronous message.

Example:

```
ATO_FUNC = ATO_PAUSE
CALL 'ATO'
ATO_FUNC = ATO_PENDING    /* only necessary, if AUTOPEND=NO */
CALL 'ATO'
ATO_FUNC = ATO_COPY_PS_TO_STRING
ATO_BUFF = COPIES(' ',28)
ATO_POSITION = D2C(1,20)
CALL 'ATO'
IF ATO_BUFF = 'Title line of the new screen' THEN
  DO
    ...
  END
```

Figure 17. Example ATO_PAUSE

5.10. ATO_PENDING

Syntax:

ATO_FUNC	**ATO_PENDING**
ATO_BUFF	

Description:

Asynchronous messages sent to a dialog can temporarily be suppressed using AUTOPEND=NO in ATO_SET_SESSION_PARAMETERS in order not to interrupt the normal dialog flow. ATO_PENDING allows to get these messages, i.e., the new virtual screen content is now available to the application program and, for example, can be copied into a buffer using ATO_COPY_PS_TO_STRING.

Note:

SAP: A special message shows that a task is changing its status into long-running. Later, when the processing is done, there appears the screen that is usually expected. This is recognized by ATO_PAUSE. Depending on the AUTOPEND parameter it is now necessary to get the new virtual screen contents using ATO_PENDING.

Example:

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

Figure 18. Example ATO_PENDING

5.11. ATO_PUTLOG

Syntax:

ATO_FUNC	**ATO_PUTLOG**
ATO_BUFF	*meldung*

Description:

ATO_PUTLOG writes the text in ATO_BUFF to ATOLOG.

Note:

VSE: ATO_PUTLOG writes to SYSLST

MVS: ATO_PUTLOG writes to the ddname ATOLOG

Also other commands like SAY in REXX are available to write messages.

Example:

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

Figure 19. Example ATO_PUTLOG

5.12. ATO_PUTWTO

Syntax:

ATO_FUNC	**ATO_PUTWTO**
ATO_BUFF	

Description:

PUTWTO writes the text in ATO_BUFF to the console.

The following figure shows an example of the ATO_PUTWTO command.

Example:

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

Figure 20. Example ATO_PUTWTO

5.13. ATO_QUERY_CURSOR

Syntax:

ATO_POSITION	
ATO_FUNC	**ATO_QUERY_CURSOR**
ATO_BUFF	

Description:

ATO_QUERY_CURSOR serves to query the cursor position. This function puts the current cursor position into ATO_POSITION looking at the whole virtual screen like one string. This allows to easily calculate the line and column as shown in the following figure.

Example:

```
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
```

Figure 21. Example ATO_QUERY_CURSOR

5.14. ATO_SEND_AID

Syntax:

ATO_FUNC	**ATO_SEND_AID**
ATO_BUFF	*key*

key **Valid values:** ENTER, CLEAR, PA1 - PA3, PF1 - PF24

This parameter defines the function key or the enter key respectively.

Default: ENTER

Description:

ATO_SEND_AID serves to send a function key to the virtual screen. Often, this function follows an ATO_SEND_KEY.

The following figure shows easy functions with ATO_SEND_AID commands that help to keep the main dialog easily readable.

Example:

```
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
```

Figure 22. ExampleATO_SEND_AID

5.15. ATO_SEND_KEY

Syntax:

ATO_FUNC	**ATO_SEND_KEY**
ATO_BUFF	*key*

key **Valid values:** Any succession of keystrokes

Description:

ATO_SEND_KEY sends any keystrokes to the virtual screen. The embedding of special keys using the escape character @ is described earlier in the manual. If the entry starts with text, the input is made at the current cursor position.

The following figure shows an example of the ATO_SEND_KEY command. @H places the cursor on the first entry field where the string *userid* is entered. Then, after @N (newline), i.e., on the first input field of the following line, *password* is entered. It is also possible to get the same effect with 2 ATO_SEND_KEY calls, which might be more readable in some circumstances. Please note that after getting the virtual screen, the cursor is automatically placed on the first input field, @H could therefore also be omitted.

Example:

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

Figure 23. Example ATO_SEND_KEY

5.16. ATO_SET_CURSOR

Syntax:

ATO_FUNC	**ATO_SET_CURSOR**
ATO_POSITION	*pos*

pos **Valid values:** Depends from the terminal defintion for the virtual screen

pos defines the new cursor position

Description:

After receiving a virtual screen, the cursor is on the first entry field. If another field should be filled, the cursor position can be changed using ATO_SET_CURSOR.

The general formula to get the cursor position is as follows:

(line-number - 1) * number of characters per line + column number

Note

Also the "virtual keystrokes" @t (tab), @n (newline) etc. allow to position the cursor on the virtual screen.

The following figure shows as example of the ATO_SET_CURSOR command how on line 24, column 5 the SAP transaction ABAP is called, assuming that the dialog is made for a model 2 with 80 characters per line.

Example:

```
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'
```

Figure 24. Example 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*

sessid **Valid values:** 1 character

This parameter defines the session.

Default: A

name **Valid values:** Max. 8 alphanumerical characters

This parameter indicates the name of the dialog to be executed.

Default: ATODLOG

netname **Valid values:** Max. 8 character terminal name according to network naming conventions.

CICS: This parameter identifies a valid network name according to CICS TCT or CICS CSD TERMINAL definitions respectively.

Default: NETATO1

applid **Valid values:** APPLID name according to network conventions.

APPLID for a ***masc-ato*** session.

CICS: This parameter identifies the application name as defined in the SIT of the corresponding CICS. This name is also displayed at the bottom right-hand edge of the screen with the transaction CEMT INQ TAS.

Default: DBDCCICS

appltry **Valid values:** 1 - 6000

appltry is the number of attempts that ***masc-ato*** executes for the establishment of a connection with the application in parameter APPLID.

The entry **APPLTRY=1** may be used to only execute a dialog when the TP monitor is active and available.

Default: 6000

autopend **Valid values:** YES, NO

autopend defines if an asynchroneously sent message, caused for example by an EXEC CICS START TRANSID or a CMSG transaction, should interrupt the normal dialg flow or if the message should be discarded for the moment.

Specifying NO allows, to process all asynchroneous messages at some well-defined points in the program.

Default: NO

lineov **Valid values:** 0 - 99999

lineov defines the number of lines per page for ATOLOG and ATOPRT.

Default: 55

logaid **Valid values:** YES or NO.

If this parameter is set to YES, the key pressed is shown in the lower right corner when following a session with LOGTERM. This helps to debug a session using LOGTERM.

If LOGTERM is not active, the value of LOGAID is ignored.

Default: NO

logterm **Valid values:** Max. 8 alphanumeric characters. Valid network name according to network conventions, NO or LOGON.

This parameter defines the network name of a display screen for the supervision and step by step reporting of the dialog. If this optional function is not required, specify LOGTERM=NO. When SUPPORT=YES is defined, specify LOGTERM=NO in order to reduce excessive output.

By using LOGTERM=LOGON, LOGTERM is dynamically activated by entering LOGON APPLID(*netname*). ***masc-ato*** then delays the processing until a LOGON APPLID(*netname*) follows.

Note:

LOGTERM is also supported for session-manager products.

The use of LOGON APPLID(...) corresponds to the default USSCMD FORMAT=PL1 in the definition USSTAB. When using FORMAT=BAL, LOGON APPLID=... has to be used. The entry follows with USSTAB message 10.

Default: NO

logtime **Valid values:** 0 - 60

logtime defines a time delay in seconds when using LOGTERM. During this time ***masc-ato*** waits with displaying the next screen.

Default: 2

logtry **Valid values:** 1 - 6000

logtry defines the number of attempts to establish a connection with the LOGTERM terminal. When using LOGTRY=1 the LOGTERM terminal is only used if available or if defined in VTAM. Otherwise the processing is continued without LOGTERM.

Default: 6000

loop **Valid values:** 1 - 100000

This parameter serves as a LOOP-control. When exceeding this value the dialog is terminated.

It is recommendable to use the programming language to detect and handle loops.

Default: 9000

mdtauto **Valid values:** YES, NO

When using MDTAUTO=YES , FILL-commands are generated automatically for MDT-display fields (eg. CICS BMS FSET fields). Fields which remain invisible to the user on the screen (Dark) and only serve for conversational prompting, may also be transmitted herewith.

When defining MDTAUTO=NO, only the fields that have been inserted or changed with FILL or MAPFLD, will be transmitted. This parameter sets the default for MAP MDTAUTO (see parameter MDTAUTO in the command MAP on page 65).

Default: YES

timeout **Valid values:** 0 - 60000

This parameter defines the max. time in seconds, while ***masc-ato*** waits for a reply. This parameter sets the default maps, ATO_PAUSE, and ATO_PENDING.

When using TIMEOUT=1440 the timeout control is disabled.

Default: 120

trace **Valid values:** NO, YES

This parameter serves to switch the ATOLOG on or off respectively. If TRACE=YES is set, all screens of the dialog are displayed.

Default: YES

support **Valid values:** NO, YES

This parameter is used for diagnostic help. With the use of SUPPORT=YES, detailed dialog evaluations are written to ATOLOG. When SUPPORT=YES is defined, you should use LOGTERM=YES for better performance.

Default: NO

VSE: When SUPPORT=YES is active, the JCL parameter //OPTION LOG,PARTDUMP has to be defined.

MVS: When SUPPORT=YES is active, the JCL parameter MSGLEVEL=(1,1) in the JOB card as well as a //SYSUDUMP DD statement has to be defined.

logmode **Valid values:** LOGMODE name according to VTAM definitions

The *logmode* parameter specifies which logmode should be applied for the virtual screen.

All screen models (model 2, 3, 4 or 5) are now supported.

Default: D4A32782

logtmod **Valid values:** LOGMODE name according to VTAM definitions

This parameter defines the logmode for the LOGTERM. Usually this should be the same logmode as specified in the *logmode* parameter.

All screen models (model 2, 3, 4 or 5) are now supported.

Default: D4A32782

Description:

The ATO_SET_SESSION_PARAMETERS command defines the start of a ATO dialog. By means of corresponding parameters, a TP monitor and a logical terminal is selected for processing the dialog. Furthermore, with the respective parameters, a detailed LOG or a LOG terminal running parallel to the dialog may be specified.

The communication between ATO_SET_SESSION_PARAMETERS and the applications or VTAM definitions is described in the "*Installation Manual*".

Example:

```
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'
```

Figure 25. Example ATO_SET_SESSION_PARAMETERS

5.18. ATO_SLEEP

Syntax:

ATO_FUNC	**ATO_SLEEP**
ATO_POSITION	*sec*
ATO_BUFF	

sec **Valid values:** 1 - 3600.

time value in seconds for the ATO_SLEEP command

Description:

ATO_SLEEP allows to interrupt a dialog for a certain amount of time. ATO_SLEEP does not read a new virtual screen.

The following figure shows an example of the ATO_SLEEP command.

Example:

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

Figure 26. Example ATO_SLEEP

5.19. ATO_TERMINATE

Syntax:

ATO_FUNC	**ATO_TERMINATE**
ATO_BUFF	

Description:

ATO_TERMINATE closes the ATO dialog and has to be programmed only once at the end of the dialog.

Before ATO_TERMINATE all sessions that are connected have to be closed using ATO_DISCONNECT_PS.

The following example shows the termination of the dialog and the exit of the program setting return-code 16.

Example:

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

Figure 27. ExampleATO_TERMINATE

6. COMPATIBILITY MODE

6.1. Internal Work Areas

The work areas WORK1 - 9 are work areas which are referenced using commands with field parameters. Each of these areas are 80 bytes long and are initialized with SPACES. They can be applied in the TO= and FROM= parameters of the corresponding commands.

The following commands address the work areas WORK1 - 9:

- **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. Examples of Application of the WORK Areas

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

Figure 28. Example 1 WORK Areas

```
          :
          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
          :
```

Figure 29. Example 2 WORK Areas

6.3. ABORT

Syntax:

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

retcode **Valid values:** 04, 08, 16

A return code that is set at the termination of a dialog. This return code may be referenced in the JCL. This way, a possible dependency of any subsequent processing can be controlled.

Default: 08

Description:

ABORT immediately terminates the dialog. Any commands following this statement will not be processed. ABORT is used to prevent any further processing after an error. How to use this statement is shown in example in figure 7.

Example:

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

Figure 30. Example ABORT

Note:

No logoff from CICS is initiated with ABORT. This command has the same effect as switching off the logical video display. An alternative and better method is provided in the dialog sequence in figure 32.

```
FATAL  MARK
       PUTPRT DATA='FATAL-ERROR'
       MAP  KEY=CLEAR
       MAPEND
       MAP LASTMAP=YES,RC=16
       FILL DATA='CSSF LOGOFF'
       MAPEND
       :
```

Figure 31. ABORT Alternative

Conventions for Return Codes:

00 - Processing successful
04 - Warning
08 - Error
16 - Serious error

6.4. COPY

Syntax:

	Command	*Parameter*
	COPY	*name*

VSE: *name*

Valid values: Valid bookname according to VSE conventions.

The copy book must be available in one of the "// LIBDEF SEARCH" libraries.

MVS: *name*

Valid values: Valid member name according to MVS conventions.

The member must be in one of the datasets of the "//SYSLIB" concatenation.

Default: None

Description:

COPY includes predefined dialog sequences into the program execution. It functionally corresponds to the "COPY" as usually found in programming languages.

Refer to page 74 for the possible use of FILL DATA=`#VAR#' for logon sequences.

An example of the COPY command is given in figure 9.

Example:

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

Figure 32. Example COPY

6.5. DCL

Syntax:

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

field **Valid Values:** Fieldname of maximal 7 alphanumeric characters

field indicates the name of a field which is defined for further processing.This name is a **mandatory** specification.

Default: none

data **Valid Values:** 80 alphanumeric characters maximum

data may contain a constant value or arbitrary alphanumeric characters. The number of characters defines the length of the field. The DATA parameter is a **mandatory specification.**

Default: none

Description:

With the command DCL (Declare) fields may be defined, which will be used with FROM/TO in further processings. The command may appear in any position in the dialog. For a more clarity of the dialog it is recommended to use the declarations in a separate paragraph at the beginning of the dialog.

An example of application is shown in figure 10.

Example:

```
         PROLOG ...
*********************************************
* DECLARATION                                *
*********************************************
MSG01    DCL  DATA='ERROR IN LOGON PROC'
CUSTNR   DCL  DATA='123.456.789'
RETC     DCL  DATA='FALSE'
         :
*********************************************
* MAIN PROCESSING                            *
*********************************************
         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
*********************************************
* PROCEDURES                                 *
*********************************************
LOGON    PROC
         :
         MOVE DATA='TRUE',TO=RETC
         PROCEND
*
ERR08    PROC
         :
         PROCEND
         EPILOG
         END
```

Figure 33. Example of DCL

Syntax:

	Command	*Parameter*
	END	

Description:

END concludes a dialog and must follow the EPILOG command immediately. See also chapter "EPILOG" on page 38.

An example of the END command is shown in figure 11.

Example:

```
PROLOG
 :
EPILOG
END
```

Figure 34. Example END

6.6. EPILOG

Syntax:

	Command	*Parameter*
	EPILOG	

Description:

EPILOG does some housekeeping tasks and is used to conclude a ATO dialog. After EPILOG, an END command has to be defined. See also chapter "END" on page 46.

An example of the EPILOG command is given in figure 12.

Example:

```
PROLOG
 :
EPILOG
END
```

Figure 35. Example EPILOG

6.7. FILL

Syntax:

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

label **Valid values:** max. 7 characters

If a *label* is assigned to a FILL command, this field may be referenced with a GETRDR or MOVE command.

data **Valid values:** max. 80 characters

This parameter contains the text that is to be transferred to the defined position.

Variables can be transferred from the USER Exit ATOEXI with the use of DATA=`#VARn#'. Also compare page 93 of chapter "User Exit ATOEXI".

CICS: The TCT parameter UCTRAN=YES causes the conversion into capital letters.

TSO: Within TSO, various conversions are active (Codepages in ISPF etc.)

line **Valid values:** 01 - 24

This parameter defines the display screen row in which the value is to be transferred.

Default: 01

column **Valid values:** 01 - 80

This parameter defines the display screen column in which the value is to be transferred.

Default: 01

Description:

FILL is used to transfer screen values to an input field. The LIN- and COL-parameters determine the input position. The input length is determined by the number of characters between the quotation marks. If the DATA-parameter extends over 2 or more lines, one has to define a continuation mark in column 72. In this case the next line has to begin at column 16. This notation is according to assembler-conventions. Compare example 2 of figure 14.

Note that a FILL command may only appear between MAP and MAPEND commands. See examples in figure 13 and 14.

Note:

CICS: Transactions like CESN check the input length. When entering transaction codes on a blank screen it is advantageous to use the command MAP DATA='*tran*'. This corresponds to an entry in line mode.

If the screen is cleared using MAP KEY=CLEAR, MAP DATA='*data*' has to be used afterwards.

Display screens with LIN/COL indicators on the bottom may be helpful in determining the values for the LIN and COL parameters.

Example:

```
MAPN    MAP
        FILL COL=5,LIN=10,DATA='INPUT'
        MAPEND
        :
```

Figure 36. Example 1 FILL

```
1...+....1....+....2....+....3....+....4         7
        0         0         0         0         2
_____________________________________________________________________________
        GETRDR EOF=EOF1,TO=LINE1
MAPN    MAP
LINE1   FILL  DATA='XXXXXXXXXXXXXXXXXXXXXXXXXXXXC
              XXXXXXXXXXXXXXXXXXXXXXXXX'
        MAPEND
        :
```

Figure 37. Example 2 FILL

6.8. GETRDR

Syntax:

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

mark **Valid values:** max. 7 characters.

This parameter defines a branch mark that will be branched to in case of an EOF condition (End of data). *mark* has to be defined with the command MARK in order to process an EOF condition.

This parameter is mandatory.

field **Valid values:** 7 characters maximum defined field label or WORK1 - 9.

This field is used to receive values with the command GETRDR. A maximum of 80 characters can be read with each GETRDR.

VSE: The values are read in via SYSRDR.

MVS: The values are read in via the ATORDR DD statement.

Default: WORK1

Description:

GETRDR is used to read input card values. If the default for a field is not used, a TO-field has to be referenced.

MVS: With the JCL definition //ATORDR DD DSN=*datasetname(member)*,DISP=SHR you may also read in PDS members.

ATORDR has a DCB definition of DCB=(LRECL=80,RECFM=FB).

Examples:

```
        :
        GETRDR TO=VAR1,EOF=EOF1
MAPN    MAP
VAR1    FILL COL=02,LIN=02,DATA='VARIABLE1'
        MAPEND
        :
EOF1    MARK
        PUTPRT DATA='END OF DATA'
        GOTO MARK=END1
        :
```

GETRDR JCL data:

```
MANDANT1
/*
```

Figure 38. Example 1 GETRDR

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

GETRDR JCL data:

```
0...+....1....+....2....
1        0         0

ITEM1       00010.0502
ITEM2       07012.7003
/*
```

Figure 39. Example 2 GETRDR

6.9. GOTO

Syntax:

	Command	*Parameter*
	GOTO	**MARK=***mark*

mark **Valid values:** max. 7 characters

This parameter defines a branch mark that is to be declared with the command MARK or a label of a MAP definition.

Description:

This command is used to branch to a defined *label*-MARK or *label*-MAP.

Example:

```
        :
        GOTO MARK=END1
        :
END1    MARK
        :
        GOTO MARK=MAP10
        :
MAP10   MAP
        :
      MAPEND
```

Figure 40. Example GOTO

Note:

Use this command carefully in order to promote the readability of the dialog for maintenance or error diagnoses. If possible try to code the dialog sequentially (top down) with as few branching statements as possible. As an alternative the command PERFORM may be used in order to execute a command sequence. The PERFORM command is described on page 56

6.10. HLLCALL

Syntax:

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

label **Valid Values :** 7 characters maximum.

With the specification of a label the contents of the DATA-parameter can be filled eg. by a MOVE command.

Default: none

data **Valid Values:** 80 alphanumeric characters maximum.

This field defines an area in which the data strings are put. It is made available to the calling program under the name HLLDATA for program control.

Default: none

field **Valid Values:** Defined field label or WORK1 - 9 with max. 7 characters.

field is a value which is made available to the calling batch program by means of the HLLTO area.

Default: WORK1

Description:

HLLCALL serves to communicate with the batch program, which the dialog has called using CALL ATOHLL. . In addition data can be made available for further processing via the area of the HLL Interface (see HLL Interface area). A renewed calling of the ATO dialog with CALL ATOHLL continues the processing after the HLLCALL-command.

Note:

HLLCALL can only be used in dialogs which have been called from a batch program by means of CALL ATOHLL. With the specification PROLOG SUPPORT=YES the transferred data areas and parameters can be logged and checked in detail. HLLCALL is called implicitly after a MAP LASTMAP=YES and after the command ABORT. After these two commands the ATO dialog is terminated and renewed calling of the dialog with the HLL interface leads to undesired results.

Example 1:

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

Figure 41. Example 1 HLLCALL

Example 2:

```
        :
L1      MARK
        HLLCALL DATA='NEXT',TO=WORK2
        PUTPRT FROM=WORK2
        SCAN   FROM=WORK2,DATA='/*',FOUND=END1
        MAP    DATA='FIBU'
        MAPEND
        MOVE   TO=CREDI,FROM=WORK2,COL=01
        MAP
CREDI   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
```

Figure 42. Example 2 HLLCALL

Example 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
```

Figure 43. Example 3 HLLCALL

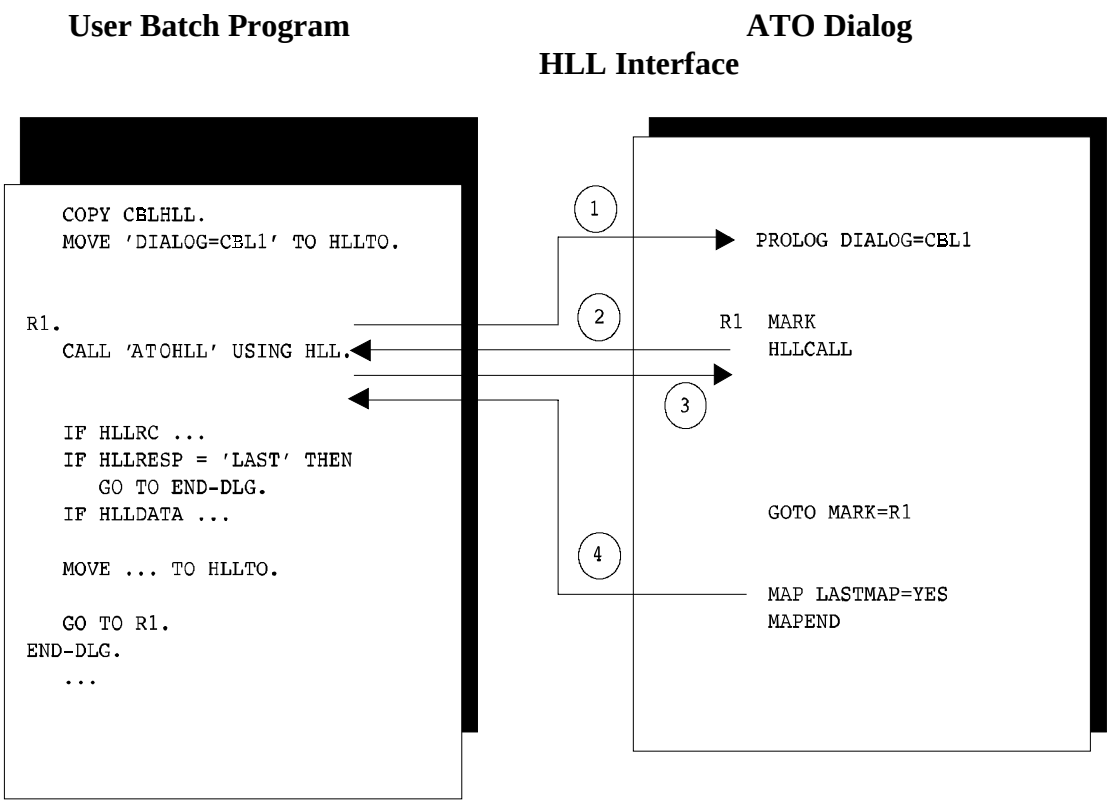
6.11. ATOHLL Interface

Description:

The ATOHLL interface serves to call a ATO dialog from a batch program and to communicate by means of HLLCALL.

The following diagram shows the principles of the ATOHLL interface and the process and communication with the ATO dialog.

The IF-sentences used in this example can also be replaced with corresponding SELECT statements.



1) The ATO dialog is initialized with the first call of ATOHLL from the batch-program (INIT-Phase).

2) The ATO dialog transfers the control by means of HLLCALL back to the batch-program, whereby information is transferred by means of HLLCALL DATA=.

3) The ATO dialog is again called by means of the transfer of data in the field HLLTO (NEXT-Phase).

The sequences 2) and 3) can be repeated as often as necessary.

4) LASTMAP=YES indicates to the batch-program the end of the ATO dialog (LAST-Phase).

6.12. HLL Interface Area

To be able to communicate from a batch-program with a ATO dialog via HLL interface, a communication area, the so called HLL interface area, is used. This area has a certain structure, which has to be filled by the batch program or is filled with the corresponding data after returning from the ATO dialog to the batch-program. The following fields are available :

Fieldname	**Usage**
HLLLENG	Length of the HLL interface area. This length is fixed at 256 Bytes.
HLLREQU	Request type. Will automatically be tracked by ATO .
HLLRESP	Response type. Will be filled with the value LAST when LASTMAP=YES.
HLLRC	Return code field. This field is filled with a corresponding return code by ATO when returning to the calling batch program.
HLLDATA	Control area for the ATO dialog with a length of 80 bytes. HLLDATA may not be altered by the batch program.
HLLTO	Transfer area for the user batch program with a length of 80 bytes.

The following figures demonstrate the coding of the HLL interface area for the programming languages ASSEMBLER, COBOL and PL/I, however, this structure can similarly adapted with other programming languages.

Assembler structure for 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
```

Figure 44. HLL Interface Area (Assembler Structure)

COBOL Structure for 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.
```

Figure 45. HLL Interface Area (COBOL Structure)

PL/I Structure for 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(' ');
```

Figure 46. HLL Interface Area (PL/I Structure)

6.13. HLL Interface Calling Technique

A user program that calls a ATO dialog (CALL ATOHLL) has to be assembled correspondingly. The following figures demonstrate the Job Control for assembly of a user program with built in ATO dialog call by means of an assembler program. Also see the notes as from page 35.

```
// 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
/*
```

Figure 47. HLL Calling Technique VSE

```
//jobname JOB ...
//STEP1   EXEC PROC=ATOGEN,DIALOG=TESTHLL
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
/*
//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=*
...
/*
```

Figure 48. HLL Calling Technique MVS

6.14. HLL Interface Instructions

Calling and Parameter Transfer

Apart from various programming languages a ATO dialog can also be called via ATOHLL with tools and vendor utilities. For detailed instructions on the calling of a dialog, see the corresponding chapters that describe the calling of assembler modules.

ATOHLL can be called from a user program in the form of

o object modules

or

o phases (VSE), load modules (MVS)

The calling of an object module is a widely used method and is supported by all popular programming languages. With this method the module ATOHLL is automatically merged into the user program at linking time. The entry point and load point of ATOHLL are identical, therefore no further entry statements have to be defined for the linkage editor.

In a user program the calling of a ATO dialog in form of a phase or load module is supported by the corresponding compiler with the corresponding options. With this method ATOHLL is not merged into the user program, but is loaded at execution time.

The parameter transfer takes place according to the standard linkage conventions and in the ASM standard format.

The ATOHLL interface expects that the length which was defined in the field HLLLENG is available as a whole. In case the length definition is wrong other areas of the user program could be overwritten by mistake.

The ATOHLL interface always sets a return code. By means of the corresponding coding this may be reviewed in the user program after returning from the ATO dialog. In the case of programming languages which do not support return codes of modules this return code is also available in the field HLLRC.

Storage Requirements

By calling a ATO dialog from a user program via ATOHLL, ***masc-ato*** aded into the same region (VSE) or the same address space (MVS) as the user program to be executed. Therefore the memory requirement is larger, namely about 256 K for ***masc-ato*** and about 64 K for explicit memory requests of ***masc-ato*** during execution time. In addition further implicit memory requests of the called interfaces have to be taken into consideration eg. VTAM, DL/I, SQL etc.

By defining large enough regions or partitions with

VSE: EXEC USERPGM,SIZE=nnnK (GETVIS for ATO and VTAM)

MVS: EXEC PGM=USERPGM,REGION=nnnK (Program size plus GETMAIN's)

you can avoid abnormal termination due to the partition size or region size being too small.

Correlation with Programming Languages

Various programming languages offer diagnostic help, eg. by localizing data exceptions by means of SPIE/STXIT. Within the ***masc-ato*** no SPIE/STXIT is used, because program checks can be initiated by the user exit ATOEXI. This condition therefore guarantees the full spectrum of diagnostic help for the user program in the selected programming language.

Internally ***masc-ato*** operates with various asynchronous constructions and with multitasking. With the PROLOG option TIMEOUT=1440 a task can be eliminated. This may be necessary under VSE since only a couple of task levels are supported.

If ***masc-ato*** discovers an error after calling a dialog, the control is returned to the user program with the corresponding return code. This enables the user program to execute the necessary steps according to each situation, eg. closing of files, rewind of VSE tapes or to return to CBLTDLI under DL/1. A corresponding return code is also set if the called ATO dialog is terminated with the command ABORT.

Please note that a ATO dialog should always be ended if possible with the sequence MAP LASTMAP=YES.

Within a batch program ***masc-ato*** can only be initiated **once**.

JCL Definitions

For the input/output-commands GETRDR, PUTLOG and PUTPRT ATO requires different allocations.

Under MVS the following DD-statements are used:

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

//SYSUDUMP DD ... (Diagnostic help)
```

Note that ***masc-ato*** does not use DD-allocations with //SYS.... This allows components, that are applied within a user program, to allocate inputs or outputs to SYSPRINT, SYSIN, SYSOUT etc., eg. SORT programs.

6.15. LOGTIME

Syntax:

	Command	*Parameter*
	LOGTIME	**SEC=***sec*

sec **Valid Values:** 0 - 60

When using PROLOG LOGTERM this parameter defines the delay time in seconds between to ATO commands being executed.

Default: 3

Description:

Using the PROLOG LOGTERM function this command serves to define the delay time between the ATO commands. LOGTIME can be coded at any arbitrary position in the ATO dialog. This is used if the value of the LOGTIME parameter in the PROLOG command has to be reset. With this command you achieve greater efficiency when checking a ATO-dialog by means of a PROLOG LOGTERM function, eg. by letting already tested dialog sequences (eg. logon's etc.) pass at a higher speed and only setting a larger interval as from a certain processing-sequence.

Example:

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

      :
      LOGTIME SEC=4
MAP50 MAP DATA='FIBU'
      MAPEND
```

Figure 49. Example LOGTIME

6.16. LOOP

Syntax:

	Command	*Parameter*
	LOOP	**COUNT=***count*

count **Valid Values:** 1 - 100000

This parameter resets the value of the PROLOG LOOP-parameter to the desired number.

Default: 9000

Description:

By using the LOOP-command at any arbitrary position in the dialog the internal LOOP counter can be set to a new value.

Example:

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

       :
       LOOP COUNT=3000
MAP50  MAP DATA='FIBU'
       MAPEND
       SCAN DATA='MAIN-MENU',FOUND=MAP60
       MAP KEY=PF3
       MAPEND
       MAP KEY=CLEAR
       MAPEND
       GOTO MARK=MAP50
MAP60  MAP
       FILL DATA='123456',LIN=07,COL=14
       MAPEND
       :
```

Figure 50. Example LOOP

6.17. MAP

Syntax:

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

sessid **Valid values:** 1 character

sessid switches to a certain session. This is necessary whenever several sessions are active in parallel. *sessid* is valid for all following MAP commands that do not specify this parameter.

Default: currently active session

data **Valid values:** 80 alphanumeric characters maximum

data may contain a constant value or arbitrary alphanumeric characters. The number of characters define the field length. If a label is allocated to the MAP-command, it is possible to reference this field with TO parameters.

Default: none

label **Valid values:** Maximum of 7 characters

label may be used as a branch address for the GETRDR, GOTO or SCAN commands or as a TO-address for filling the data of the DATA parameter.

Default: none

line **Valid values:** 01 - 43

This parameter defines the row where the cursor on the input screen will be positioned with data entry or with the pressing of a PF key.

Default: 01

column **Valid values:** 01 - 132

This parameter defines the column where the cursor will be positioned on the input screen when pressing the enter or PF key.

Default: 01

retcode **Valid values:** 00, 04, 08, 16

A job control return code can be set with this parameter. Also see "Conventions for Return Codes" on page 23.

Default: 00

lastmap **Valid values:** NO, YES, PASS

The statement LASTMAP=YES informs ***masc-ato*** that it relates to the last MAP instruction to be processed.

PASS is used to change from a session with an APPLID to another. This functionality can now be achieved easier using several parallel session using the SESSBEG, SESSMOD/SESSSET, and SESSEND commands.

Default: NO

mdtauto **Valid values:** NO, YES

When setting MDTAUTO=NO only fields that are transferred with FILL or MAPFLD are transmitted.

Default: See PROLOG MDTAUTO on page 60.

The default is set with PROLOG MDTAUTO.

timeout **Valid values:** 1 - 60000

With the statement TIMEOUT=*timeout* the maximum duration is specified (in seconds) which the dialog will wait for an answer from the TP monitor. When this value is exceeded the dialog will be aborted.

The default value is set with the PROLOG TIMEOUT parameter.

Default: see PROLOG TIMEOUT-parameter on page 21.

SAP: For ABAP/TF70/TF78/TK31 this value has to be increased.

MVS: For long running transactions the TIME-parameter has to be adjusted in the JOB-statement or EXEC-statement (eg. TIME=1440) or a corresponding job class has to be chosen for processing of the dialog.

key **Valid values:** ENT (Enter), CLEAR, PA1 - PA3, PF1 - PF24

With this parameter the function key necessary for this transaction is specified.

Default: ENT (Enter)

PA1-PA3 are supported in transaction mode only.

When using the CLEAR.key to delete a screen, the next command sequence to be used for displaying data on the unformatted screen is MAP DATA='*data*'.

CICS: PA1 is not supported for TCT-PRINTO/CSD-TERMINAL PRINTER()

SAP: It is to your advantage to use the function execution Nxxx on line/column 24/5 instead of the predefined function keys for a better overview.

MVS: The CLEAR-key may not be used for TSO-applications. PA1 is used in TSO for interrupting the output on the screen after a TSO command.

receive **Valid values:** YES, ONLY

With RECEIVE=ONLY a second screen can be received from the TP-monitor (see description for PENDING as from page 52).

Default: YES

Description:

The command MAP is used to position the cursor on the input screen and to define the operational function keys.

Between MAP and MAPEND only FILL and MAPFLD commands may be used.

Example:

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

Figure 51. Example MAP

```
         :
*********************************************
* SESSION PASSING WITH NEW APPLID             *
*********************************************
         MAP DATA='CECI IS PA LU(CICS2)'
         MAPEND
         MAP KEY=ENT
         MAPEND
         MAP KEY=PF3,LASTMAP=PASS
         MAPEND
*********************************************
* NEW APPLID-SESSION                         *
*********************************************
         MAP KEY=CLEAR
         MAPEND
         MAP DATA='CESN'
         MAPEND
         MAP
         FILL LIN=..,COL=..,DATA='#VAR1#'  USERID
         FILL LIN=..,COL=..,DATA='#VAR2#'  PASSWORD
         MAPEND
         :
```

Figure 52. Example MAP LASTMAP=PASS Example

6.18. MAPEND

Syntax:

	Command	*Parameter*
	MAPEND	

Description:

The MAP instruction is concluded with the MAPEND command and an answer of the TP monitor is awaited. The content of the display screen received can be processed, e.g. with a SCAN or MOVE command.

Example:

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

Figure 53. Example MAPEND

6.19. MAPFLD

Syntax:

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

label **Valid values:** max. 7 characters

label can be referenced with a TO parameter.

Default: None

data **Valid values:** max. 80 characters

This parameter defines the value that should be assigned to a field.

CICS: The TCT parameter UCTRAN=YES causes the conversion into capital letters.

TSO: Within TSO various conversions may be active (Codepages in ISPF etc.)

Description:

MAPFLD transfers the value in the DATA=-parameter of a MAP, where the length equals the number of characters between the quotation marks. The MAPFLD value may be filled with the GETRDR or MOVE commands. Figure 31 and 32 on page 71 show the use of this command.

MAPFLD commands can be coded only following a FILL command between MAP and MAPEND commands. Generally, this command is used to fill fields, declared with a MAP-FILL statement, with values of variable content.

Examples:

```
         :
         SCAN DATA='FOLDER'
         MOVE COL=6,TO=FNAME1
         :
MAPN     MAP
         FILL LIN=14,COL=07,DATA='REPRODUCTION OF'
FNAME1   MAPFLD DATA='XXXXXXXX'
         MAPFLD DATA=' HAS ENDED'
         MAPEND
         :
EOF1     MARK
         :
```

Figure 54. Example 1 MAPFLD

This example enters the value "REPRODUCTION OF" on line 14 beginning at column 7. With the first MAPFLD command a variable value of 8 characters is added. Because the label FNAME1 precedes the first MAPFLD command, the DATA parameter defines the length of this receiving field and is filled accordingly with the GETRDR or MOVE command. The second MAPFLD instruction adds the value "HAS ENDED" for the conclusion of the line, which results to the final line value of "REPRODUCTION OF XXXXXXXX HAS ENDED".

```
         :
         MAP  KEY=CLEAR
         MAPEND
         MOVE TO=FOLDER,DATA='FOLDER1'
MAPN     MAP  DATA='SBDC S='
FOLDER   MAPFLD DATA='XXXXXX'
         MAPEND
         :
```

Figure 55. Example 2 MAPFLD

6.20. MAPFILL

Syntax:

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

data **Valid Values:** Up to 1 line of alpanumerical characters (80 or 132)

data can contain a constant value or any alphanumeric characters. The number of characters determines the length of the line.

Default: None

line **Valid Values:** 01 - 43

This parameter defines the line where to put the entry.

Default: 01

column **Valid values:** 01 - 132

This parameter defines the line where to put the entry.

Default: 01

sessid **Gültige Werte:** 1 character

This parameter specifies the session where to direct the data entry.

Default: Currently active session

Description:

The command MAPFILL makes a data entry on a formatted screen.

Example:

```
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
        :
```

Figure 56.Example MAP

6.21. MAPKEYS

Syntax:

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

label **Valid values:** Maximum of 7 characters

label may be used as a branch address for the GETRDR, GOTO or SCAN commands or as a TO-address for filling the data of the DATA parameter.

Default: none

data **Valid values:** up to 80 characters

data contains the text that is to be sent to the screen.

Imbedded into the text, also cursor control characters may be used. If at the beginning none of these is specified, the text starts at the actual cursor position.

Using DATA='#VARn#' variables from the user exit ATOEXI may be used. Please find the details on page **Fehler! Textmarke nicht definiert.**.

CICS: The use of the TCT-UCTRAN=YES parameter causes the translation of all characters into uppercase

TSO: Within TSO various translations are active like codepages in ISPF, etc.

Description:

MAPKEYS serves to send keystrokests an entry field starting at the current cursor position. The length of the entry is determined by the number of characters between the apostrophes minus the cursor control keys marked by @. If the *data* parameter occupies more than two lines, a continuation mark has to be placed in column 72 and the next line has to start at column 16. The notation is according to assembler rules.

Please note that MAPKEYS commands may only be used in between of a MAP and MAPEND command.

Instead of the MAPKEYS command it is also possible to use FILL where the entry has to placed onto the virtual screen specifying the exact line and column.

Bemerkung:

CICS: Some transactions like CESN check the lenth of the input. Entering transaction codes on an empty screen, it is recommendable to use MAP DATA='tran'. This is like an entry in line mode.

Example:

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

Figure 57. Beispiel MAPKEYS

The example above places the string 'USERID' to the actual cursor position, then the @N causes a *newline* in order to enter 'PASSWORD' on the first input field of a new line.

6.22. MARK

Syntax:

	Command	*Parameter*
label	**MARK**	

label **Valid values:** 7 characters maximum

label defines a branch mark and is a **mandatory entry**.

Default: None

Description:

MARK is used to define a branch mark which may be referenced in the commands GETRDR, GOTO, SCAN, SCANB and PENDING.

ATO internally maintains a loop counter for MARK. The obvious reason is to prevent an endless loop. This loop counter can be set to a maximum value in the PROLOG command using the LOOP parameter.

Example:

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

Figure 58. Example MARK

6.23. MOVE

Syntax:

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

data **Valid values:** Max. 80 characters

This parameter contains the data which should be transferred to a field.

The length of the data to be transferred is determined by the number of characters between the quotes.

fromfield **Valid values:** WORK1 - 9, SCREEN, SPACES or a label

This fieldname defines the area from which the values are transferred.

Default: WORK1

column **Valid values:** 01 - 80

This parameter defines the display screen column of WORK1 - 9 and SCREEN, from which the value should be transferred.

Default: 01

line **Valid values** 01 - 24

This parameter defines the line from which the value has to be transferred from **SCREEN** .

Default: 01

tofield **Valid values:** Fieldname with max. 7 characters, WORK1 - 9 or a label.

This fieldname defines an area into which values are transferred.

Default: WORK1

tocolumn **Valid values:** 01 - 80

This value defines the column of WORK1 - 9, into which will be transferred.

Default: 1

tolength **Valid values:** 01 - 80

This value defines the length of the data in WORK1 - 9, which is to be transferred.

Default: 80

Description:

MOVE serves for the transfer of various areas. With WORK1 - 9 / SCREEN the parameters TOLEN, TOCOL, LIN and COL can be defined additionally.

Examples:

```
         :
MAP1     MAP  DATA='TRA1'
         MAPEND
         SCAN DATA='FOLDER',NFOUND=NF1
         MOVE FROM=SCREEN,TO=FNAME1
         :
         MOVE FROM=SCREEN,COL=8,LIN=3,TO=FNUMM1
         :
MAP2     MAP
         FILL LIN=14,COL=07,DATA='REPRODUCTION OF'
FNAME1   MAPFLD DATA='...FOLDER...'
         MAPFLD DATA=' HAS ENDED, NO='
FNUMM1   MAPFLD DATA='000000'
         MAPEND
         :
NF1      MARK
         :
```

Figure 59. Example 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=AMNT
        MOVE FROM=WORK2,COL=21,TO=ITEM
        :
B20     MAP
TEXT    FILL LINE=10,COL=14,DATA='XXXXXXXXXXX'
AMNT    FILL LINE=11,COL=20,DATA='XXXXX.XX'
ITEM    FILL LINE=12,COL=10,DATA='XX'
        MAPEND
        :
        GOTO MARK=L20
END1    MARK
        :
```

Figure 60. Example 2 MOVE

6.24. PENDING

Syntax:

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

label **Valid values:** 7 characters maximum

label is a branch mark defined with *label*-MARK or *label*-MAP for the FOUND- or NFOUND-condition.

sec **Valid values:** 1 - 60000

See TIMEOUT parameter in chapter "MAP" on page 65.

Default: 60

In the case where a TP monitor-transaction **always** transmits a second screen-output at the same position of the transaction, as an alternative to PENDING the command MAP RECEIVE=ONLY may also be used.

CICS: PENDING is used for receiving extraordinary messages (e.g. CMSG).

SAP: PENDING is used for the reproduction of folders via SBDC or TF70 / ABAP.

Description:

masc-ato strictly operates according to the dialog cycles of in- and outputs which are processed by the commands MAP/MAPEND.

Except for the first transaction, ***masc-ato*** always determines which transaction should be processed next. Exceptions are the commands PENDING and MAP RECEIVE=ONLY.

If the TP monitor starts a transaction, it will collide with the transaction start of MAP/MAPEND and will be rejected by ***masc-ato***. TP monitors like CICS try to repeat the pending task at each transaction end, which leads to a repeated collision and rejection by ***masc-ato***.

If a TP monitor transaction is pending that has been rejected by ***masc-ato***, it can be initiated with the use of the PENDING command. PENDING signals the TP monitor to start the pending transaction.

If the PENDING-FOUND condition is true, the TP monitor has started the pending transaction. Hereby ***masc-ato*** functions like after a MAPEND, ie. the screen is ready for further processing.
However, if the PENDING-NFOUND condition is true, ***masc-ato*** acts just like a GOTO, where the processing continues at the position defined in the NFOUND-parameter.

CICS: When a ATO dialog is terminated, it is the responsibility of CICS to handle any pending transactions. CICS usually restarts these transactions instead of the expected "Good Morning" transaction. If the same or another dialog is executed at the same logical terminal, a violation of the security level or other undesired conditions are likely to be expected, if the ATO dialog has not been coded to handle such restart situations.

SAP: With long running transactions, e.g. the reproduction of folders via SBDC, the "Finished" message will be initiated by a CICS START Transaction. It can take several hours for this message to appear, and for this reason the collision has to be initiated by the ATO dialog. With this in mind, go through the examples in figure 36 and 37 on page 82.

Note:

PENDING is used chiefly to capture the messages and transactions of TP monitors outside the dialog cycles. Pending transactions can be started by CICS applications, e.g. EXEC CICS START TRANSID or by means of the CMSG transaction.

As these "suspended" transactions are always connected to a specific ***masc-ato*** terminal, it would be an advantage to use a reserved ***masc-ato*** terminal for each ***masc-ato*** application in order to prevent conflicts with previously aborted dialogs.

For SAP applications it is recommended to additionally use a reserved SAP user per ***masc-ato*** terminal.

Example:

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

Figure 61. Example 1 PENDING CICS-SAP

```
          :
          MAP DATA='TRA1'
          MAPEND
          SCAN DATA='MESSAGE OF TRA1'
          :
          PENDING NFOUND=NTRA2
          SCAN DATA='MESSAGE OF TRA2'
          :
NTRA2     MARK
          :
END       MARK
          MAP KEY=PF3
          MAPEND
          PENDING
          MAP KEY=CLEAR
          MAPEND
          MAP LASTMAP=YES,DATA='CESF LOGOFF'
          MAPEND
          :
```

Figure 62. Example 2 PENDING

```
         :
B1       MAP  DATA='CEMT INQ TAS'
         MAPEND
B2       MAP KEY=ENT
         MAPEND
*********************************************
B3       MAP KEY=ENT
         MAPEND
B4       MAP KEY=PF3
         MAPEND
*********************************************
*     The     next     transaction     will     collide     with     CMSG          *
*********************************************
B5       MAP
         FILL DATA='CMSG'
         MAPEND
*********************************************
* Get any pending transaction                *
*********************************************
         PENDING
*********************************************
* Any pending transaction received ?         *
*********************************************
         SCAN DATA='WHICH MESSAGE'
         :
```

Figure 63. Example 3 PENDING CICS

6.25. PERFORM

Syntax:

	Command	*Parameter*
	PERFORM	**PROC=***procname*

procname **Valid values:** Max. 7 alphanumeric characters.

This parameter designates the name of the procedure to be executed.

Default: none

Description:

The command PERFORM is used for the execution of procedures. Command-sequences that are coded in the procedure *procname*, can be executed with PERFORM. The called procedure must be defined with the command PROC.

For more information see command PROC / PROCEND on pages 86 and 87.

Example:

```
          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
***********************************************
* PROCEDURE READ                              *
***********************************************
READ      PROC
READ10    MARK
          GETRDR TO=LINE,EOF=EOF99
MAP10     MAP
LINE      FILL DATA='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXC
               XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
          MAPEND
          GOTO MARK=READ10
EOF99     MARK
          PROCEND
***********************************************
* PROCEDURE FOR ERROR HANDLING                *
***********************************************
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
```

Figure 64. Example PERFORM and PROC

6.26. PROC

Syntax:

	Command	*Parameter*
Procname	**PROC**	

procname **Valid values:** Max. 7 alphanumeric characters.

procname defines the name of the procedure and is a **compulsory** entry. Up to 32 levels may be nested as a maximum.

Default: none

Description:

The command PROC marks the beginning of a procedure. The contents of such a procedure are normal dialog command-sequences, which can be called with PERFORM PROC=*procname*. The procedure ends with the command PROCEND. Within PROC / PROCEND further PROC / PROCEND and PERFORM commands may be used. A procedure may also be passed sequentially without a previous calling of PERFORM.

Important:

Within a procedure no labels may be referenced or jumped to, that lie outside PROC / PROCEND.

For a better overview we recommend that you place the procedures after the definition MAP-LASTMAP=YES.

You can find an example of PROC in figure 39 on page 85.

6.27. PROCEND

Syntax:

	Command	*Parameter*
[*label*]	**PROCEND**	

label **Valid values:** Max. 7 alphanumeric characters.

label defines a branch mark which can be branched to within a procedure.

Default: none

Description:

With the command PROCEND the end of a procedure is marked. Control goes back to the instruction that follows the calling PERFORM command. With PROCEND no return occurs if a command-sequence PROC / PROCEND is passed sequentially, ie. without previous calling by perform.

An example of PROC is found in figure 39 on page 85.

6.28. PROLOG

Syntax:

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

name **Valid values:** Max. 8 alphanumerical characters

This parameter indicates the name of the dialog to be executed.

Default: ATODLOG

netname **Valid values:** Max. 8 character terminal name according to network naming conventions.

CICS: This parameter identifies a valid network name according to CICS TCT or CICS CSD TERMINAL definitions respectively.

Default: NETATO1

applid **Valid values:** APPLID name according to network conventions.

APPLID for a ***masc-ato*** session.

CICS: This parameter identifies the application name as defined in the SIT of the corresponding CICS. This name is also displayed at the bottom right-hand edge of the screen with the transaction CEMT INQ TAS.

Default: DBDCCICS

appltry **Valid values:** 1 - 6000

appltry is the number of attempts that ***masc-ato*** executes for the establishment of a connection with the application in parameter APPLID.

The entry **APPLTRY=1** may be used to only execute a dialog when the TP monitor is active and available.

Default: 6000

lineov **Valid values:** 0 - 80

lineov defines the number of lines per page for ATOLOG and ATOPRT.

Default: 55

logterm **Valid values:** Max. 8 alphanumeric characters. Valid network name according to network conventions or NO or LOGON.

This parameter defines the network name of a display screen for the supervision and step by step reporting of the dialog. If this optional function is not required, specify LOGTERM=NO. When SUPPORT=YES is defined, specify LOGTERM=NO in order to reduce excessive output.

By using LOGTERM=LOGON, LOGTERM is dynamically activated by entering LOGON APPLID(*netname*). ***masc-ato*** then delays the processing until a LOGON APPLID(*netname*) follows.

Note:

LOGTERM is also supported for session-manager products.

The use of LOGON APPLID(...) corresponds to the default USSCMD FORMAT=PL1 in the definition USSTAB. When using FORMAT=BAL, LOGON APPLID=... has to be used. The entry follows with USSTAB message 10.

Default: NO

logtime **Valid values:** 0- 60

logtime defines a time delay in seconds when using LOGTERM. During this time ***masc-ato*** waits with displaying the next screen.
With the command LOGTIME the time value in the dialog may be reset and redefined (see command LOGTIME on page 63).

Default: 2

logtry **Valid values:** 1 - 6000

logtry defines the number of attempts to establish a connection with the LOGTERM terminal. When using LOGTRY=1 the LOGTERM terminal is only used if available or if defined in VTAM. Otherwise the processing is continued without LOGTERM.

Default: 6000

logaid **Valid values:** YES or NO.

If this parameter is set to YES, the key pressed is shown in the lower right corner when following a session with LOGTERM. This helps to debug a session using LOGTERM.

If LOGTERM is not active, the value of LOGAID is ignored.

Default: NO

loop **Valid values:** 1 - 100000

This parameter serves as a LOOP-control. When exceeding this value the dialog is terminated.

Default: 9000

mlog **Valid values:** YES, NO

If this parameter is set to YES, all macros and labels passed during the dialog are written to ATOPRT. This helps in the case of errors to see the exact flow of the dialog.

Default: NO

mdtauto **Valid values:** YES, NO

When using MDTAUTO=YES , FILL-commands are generated automatically for MDT-display fields (eg. CICS BMS FSET fields). Fields which remain invisible to the user on the screen (Dark) and only serve for conversational prompting, may also be transmitted herewith.

When defining MDTAUTO=NO, only the fields that have been inserted or changed with FILL or MAPFLD, will be transmitted. This parameter sets the default for MAP MDTAUTO (see parameter MDTAUTO in the command MAP on page 65).

Default: YES

timeout **Valid values:** 0 - 60000

This parameter defines the max. time in seconds, while ***masc-ato*** waits for a reply. This parameter sets the default for MAP- and PENDING-TIMEOUT. The default value is sufficient for most applications.

When using TIMEOUT=1440 the timeout control is disabled. TIMEOUT=1440 should only be used for long running dialogs with HLLCALL's.

Default: 120

trace **Valid values:** NO, YES

This parameter serves to switch the ATOLOG on or off respectively. If TRACE=YES is set, all screens of the dialog are displayed.

Default: YES

sessmax **Valid values:** 1 - 9

This parameter specifies the maximum number of sessions that can be simultaneously active in a dialog.

Default: 1

support **Valid values:** NO, YES

This parameter is used for diagnostic help. With the use of SUPPORT=YES, detailed dialog evaluations are written to ATOLOG. When SUPPORT=YES is defined, you should use LOGTERM=YES for better performance.

Default: NO

VSE: When SUPPORT=YES is active, the JCL parameter //OPTION LOG,PARTDUMP has to be defined.

MVS: When SUPPORT=YES is active, the JCL parameter MSGLEVEL=(1,1) in the JOB card as well as a //SYSUDUMP DD statement has to be defined.

logmode **Valid values:** LOGMODE name according to VTAM definitions

The *logmode* parameter specifies which logmode should be applied for the virtual screen.

All screen models (model 2, 3, 4 or 5) are now supported.

Default: D4A32782

logtmod **Valid values:** LOGMODE name according to VTAM definitions

This parameter defines the logmode for the LOGTERM. Usually this should be the same logmode as specified in the *logmode* parameter.

All screen models (model 2, 3, 4 or 5) are now supported.

Default: D4A32782

Description:

The PROLOG command defines the start of a ATO dialog and is therefore the **mandatory** first ***masc-ato*** instruction to be processed. By means of corresponding parameters, a TP monitor and a logical terminal is selected for processing the dialog. Furthermore, with the respective parameters, a detailed LOG or a LOG terminal running parallel to the dialog may be specified, which is used likewise for diagnostics in the support center of the licensor.

The communication between the PROLOG and the applications or VTAM definitions is described in the "*Installation Manual*".

Note:

The session startet with PROLOG contains implicitely a SESSSET and SESSBEG for session A. The session started with PROLOG does not have to end with SESSEND.

In addition to the setting of a response time, the PROLOG TIMEOUT parameter also determines the intervals for ***masc-ato*** console messages in seconds.

If a longer response time is expected for a MAP input as defined in the PROLOG TIMEOUT parameter, the MAP resp. PENDING TIMEOUT parameter has to be set accordingly. MAP TIMEOUT locally sets the time value for this entry.

The PROLOG TIMEOUT value should not be increased unnecessarily, because ***masc-ato*** will not be able to recognize any internal logical errors. For separate entries where long response times are expected the MAP TIMEOUT-parameter should be used.

The PROLOG APPLTRY parameter is used to execute a dialog only when the TP-monitor is available. For this purpose define APPLTRY=1.

Example:

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

Figure 65. Example PROLOG

6.29. PUTLOG

Syntax:

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

data **Valid values:** Max. 64 alphanumerical characters

The string *data* is written to ATOLOG. If this parameter is omitted, the data in the current WORK area of the last SCAN/SCANB or GETRDR command is listed.

Default: none

field **Valid values:** WORK1 -9 or a predefined field (eg. with DCL, MAP, FILL, SCAN(B), MAPFLD etc.)

The contents of field are output on ATOLOG.

Default: WORK1

Description:

PUTLOG either writes the text entered in the DATA parameter or the contents of the field indicated with FROM to ATOLOG. Only ONE of the two parameters can be defined. If no parameter is supplied, the default is PUTLOG FROM=WORK1.

Note:

VSE: PUTLOG writes to SYSLST.

MVS: PUTLOG writes to //ATOLOG DD SYSOUT=*.
ATOLOG has a DCB definition of DCB=(LRECL=121,BLKSIZE=121,RECFM=FBA).

Messages that only serve for dialog checking, i.e. purely informational messages, should be written by use of the PUTPRT command. See chapter "PUTPRT" on page 94.

Example:

```
MSG01   DCL DATA='ERROR: MAIN-MENU NOT FOUND'
        :
MAP1    MAP  DATA='MENU'
        MAPEND
        SCAN DATA='MAIN-MENU',NFOUND=ERROR
        :
ERROR   MARK
        PUTLOG FROM=MSG01
        :
```

Figure 66. Example PUTLOG

6.30. PUTPRT

Syntax:

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

label **Valid values:** Max. 7 alphanumeric characters

label may used as the address in a GOTO, however, it may not be referenced by a command with a TO / FROM parameter (e.g. MOVE).

data **Valid values:** Max. 64 alphanumerical characters

See command PUTLOG on page 93.

Default: none

field **Valid values:** WORK1 - 9 or predefined field (eg. with DCL,MAP, FILL, SCAN(B), MAPFLD etc.)

The contents of *field* are output on ATOPRT.

Default: WORK1

titlenr **Valid values:** 1, 2, 3

This parameter defines headings with the text of the DATA parameter. In this case a heading is unambiguously identified with the TITLE parameter. Omit this parameter, if you do not intend to write a title heading.

Default: blank

Description:

PUTPRT either writes the text supplied with the DATA parameter or the contents of the field defined with FROM to ATOPRT. This simplifies the checking of the output, e.g. for the staff of the DP Center. Only **one** of the two parameters may be defined. If no parameter is supplied, the default is PUTPRT FROM=WORK1.

Note:

VSE: PUTPRT writes to SYS001 and may be set with e.g. ASSGN SYS001,01E to 01E. The virtual printer has to be referenced with POWER statements.

MVS: PUTPRT writes to //ATOPRT DD SYSOUT=*.
ATOPRT has a DCB definition of
DCB=(LRECL=121,BLKSIZE=121,RECFM=FBA).

Example:

```
      PUTPRT TITLE=1,DATA='H E A D I N G  O N E'
      PUTPRT TITLE=2,DATA='--------------------'
      PUTPRT TITLE=3,DATA='                    '
      :
      PUTPRT DATA='LOGON SUCCESSFUL'
      :
MSG01 DCL DATA='FOLDERS NOT FOUND'
      PUTPRT FROM=MSG01
      :
```

Figure 67. Example PUTPRT

6.31. PUTWTO

Syntax:

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

data **Valid values:** Max. 64 alphanumerical characters

See command PUTLOG on page 93.

Default: none

field **Valid values:** WORK1 - 9 or predefined field (eg. with DCL, MAP, FILL, SCAN(B), MAPFLD etc.)

The contents of field are displayed on the console.

Default: WORK1

Description:

PUTWTO writes the text supplied with the DATA parameter or the contents of the field defined with FROM to the console. Only one of the two parameters can be defined. If no parameter is supplied, the default is PUTWTO FROM=WORK1.

Note:

MVS: The message sent to the console has a Routecode 11 and Descriptor Code 7.

Example:

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

Figure 68. Example PUTWTO

6.32. SCAN

Syntax:

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

label **Valid values:** Max. 7 alphanumeric characters

label may be referenced by a command with a TO / FROM parameter (e.g. MOVE).

data **Valid values:** Max. 132 characters

The defined value of *data* is searched for in the received screen content.

from **Valid values:** SCREEN, WORK1 - 9 or predefined field.

from defines an area in which the string is searched for in the DATA-parameter.

Default: SCREEN (display screen)

to **Valid values:** SCREEN, WORK1 - 9 or predefined field.

to defines an area into which the next 80 characters from the found string in the DATA parameter are set, or data that lie within the position defined by ENDLIN and ENDCOL. With WORK1 - 9, 80 characters are moved.

Default: WORK1

lin **Valid values:** 01 - 43

lin defines the line position by a SCAN on the SCREEN.

Default: 01

col **Valid values:** 01 - 132

col defines the column position from which the SCAN will begin on SCREEN or on WORK1 - 9.

Default: 01

endlin **Valid values:** 01 - 43

With *endlin* an endposition of a line may be defined for SCAN on SCREEN. The string in the DATA parameter has to be within the lines , which are defined with LIN and ENDLIN, with specification of ENDLIN. If this does not apply, the NFOUND-condition is set. The value *endlin* must be **greater** or **equal** to the value in LIN.

If only LIN is specified whereas ENDLIN does not contain a value, the string is searched on line *lin.*

Default: 01

endcol **Valid values:** 01 - 132

With *endcol* an endposition of a column or WORK1 - 9 can be defined for SCAN on SCREEN. The string in the DATA parameter has to lie within a column, that lies between COL and ENDCOL, when specifying ENDCOL. If this does not apply, the NFOUND-condition is set. The value *endcol* must be **greater** or **equal** to the value in COL. If *col* and *endcol* have the same value, the string is searched starting in *col* with the length of string.

Default: 01

toffset **Valid values:** 01 - 132

toffset defines the position where to place a field.

Default: 01

mark **Valid values:** Max. 7 characters branch mark (MARK / MAP)

mark defines a branch mark that is branched to after a FOUND or NFOUND condition of the defined value in the *data* .

Description:

SCAN searches the received content of a display screen (SCREEN) or a predefined field (WORK1 - 9, DCL-field) for a specific value defined by DATA='*data*'. In addition, the data is converted into capital letters in the internal buffer before the search. By means of the FOUND or NFOUND parameters, you can branch correspondingly to a branch mark for

further processing. In the case of a FOUND condition, the value stands left justified in the area defined with the TO-parameter (Default: WORK1) and may be processed with the commands MOVE, PUTLOG, etc.

Examples:

```
     :
     SCAN DATA='SIGN-ON IS COMPLETE',FOUND=OK1
     PUTPRT DATA='SIGN-ON ERROR'
     GOTO MARK=END1
     :
OK1  MARK
     :
END1 MARK
     :
```

Figure 69. Example 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='NOT FOUND'
       :
OK1    MARK
       :
```

Figure 70. Example 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='OF'
       :
       PROCEND
       :
```

Figure 71. Example 3 SCAN

6.33. SCANB

Syntax:

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

label **Valid values:** Max. 7 alphanumeric characters

label may be referenced by a command with a TO / FROM parameter (e.g. MOVE).

data **Valid values:** Max. 132 characters

The defined value of *data* is searched for in the received screen content.

from **Valid values:** SCREEN, WORK1 - 9 or predefined field.

from defines an area in which the string is searched for in the DATA-parameter.

Default: SCREEN (display screen)

to **Valid values:** SCREEN, WORK1 - 9 or predefined field.

to defines an area into which the next 80 characters from the found string in the DATA parameter are set, or data that lie within the position defined by ENDLIN and ENDCOL. With WORK1 - 9, 80 characters are moved.

Default: WORK1

lin **Valid values:** 01 - 43

lin defines the line position by a SCANB on the SCREEN.

Default: 01

col **Valid values:** 01 - 132

col defines the column position from which the SCANB will begin on SCREEN or on WORK1 - 9.

Default: 01

endlin **Valid values:** 01 - 43

With *endlin* an endposition of a line may be defined for SCANB on SCREEN. The string in the DATA parameter has to be within the lines , which are defined with LIN and ENDLIN, with specification of ENDLIN. If this does not apply, the NFOUND-condition is set. The value *endlin* must be **greater** or **equal** to the value in LIN.

Default: 01

endcol **Valid values:** 01 - 143

With *endcol* an endposition of a column or WORK1 - 9 can be defined for SCANB on SCREEN. The string in the DATA parameter has to lie within a column, that lies between COL and ENDCOL, when specifying ENDCOL. If this does not apply, the NFOUND-condition is set. The value *endcol* must be **greater** or **equal** to the value in COL.

Default: 01

toffset **Valid values:** 01 - 132

toffset defines the position where to place a field.

Default: 01

mark **Valid values:** Max. 7 characters branch mark (MARK / MAP)

mark defines a branch mark that is branched to after a FOUND or NFOUND condition of the defined value in the *data* .

Description:

SCANB searches **backwards** the received content of a display screen (SCREEN) or a predefined field (WORK1 - 9, DCL-field) for a specific value defined by DATA='*data*' (see also description for command SCAN on page 97).

For examples refer to SCAN command on page 104.

SAP: SCANB may be used for SBDC transactions to position the last folder on a page.

6.34. SESSBEG

Syntax:

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

sessid **Valid values:** 1 character

sessid shows for which session the SESSBEG is valid

Default: A

Description:

The SESSBEG command defines the start of an ATO session. Previously the session parameters have to be set using SESSSET.

Examples:

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

Figure 72. Example SESSBEG

6.35. SESSEND

Syntax:

	Command	*Parameter*
	SESSEND	[**SESSID=***sessid*]

sessid **Valid values:** 1 character

sessid shows for which session the SESSBEG is valid

Default: Currently active session

Description:

The SESSEND command defines theendt of an singleATO session. Before the EPILOG command all ATO sessions have to be terminated using SESSEND.

Examples:

```
SESSEND SESSID=A,
:
```

Figure 73. Example SESSEND

6.36. SESSSET / SESSMOD

Syntax:

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

name **Valid values:** Max. 8 alphanumerical characters

This parameter indicates the name of the dialog to be executed.

Default: Value from PROLOG or previous SESSET command

netname **Valid values:** Max. 8 character terminal name according to network naming conventions.

CICS: This parameter identifies a valid network name according to CICS TCT or CICS CSD TERMINAL definitions respectively.

Default: Value from PROLOG or previous SESSET command

applid **Valid values:** APPLID name according to network conventions.

APPLID for a ***masc-ato*** session.

CICS: This parameter identifies the application name as defined in the SIT of the corresponding CICS. This name is also displayed at the bottom right-hand edge of the screen with the transaction CEMT INQ TAS.

Default: Value from PROLOG or previous SESSET command

appltry **Valid values:** 1 - 6000

appltry is the number of attempts that ***masc-ato*** executes for the establishment of a connection with the application in parameter APPLID.

The entry **APPLTRY=1** may be used to only execute a dialog when the TP monitor is active and available.

Default: Value from PROLOG or previous SESSET command

lineov **Valid values:** 0 - 80

lineov defines the number of lines per page for ATOLOG and ATOPRT.

Default: Value from PROLOG or previous SESSET command

logterm **Valid values:** Max. 8 alphanumeric characters. Valid network name according to network conventions or NO or LOGON.

This parameter defines the network name of a display screen for the supervision and step by step reporting of the dialog. If this optional function is not required, specify LOGTERM=NO. When SUPPORT=YES is defined, specify LOGTERM=NO in order to reduce excessive output.

By using LOGTERM=LOGON, LOGTERM is dynamically activated by entering LOGON APPLID(*netname*). ***masc-ato*** then delays the processing until a LOGON APPLID(*netname*) follows.

Note:

LOGTERM is also supported for session-manager products.

The use of LOGON APPLID(...) corresponds to the default USSCMD FORMAT=PL1 in the definition USSTAB. When using FORMAT=BAL, LOGON APPLID=... has to be used. The entry follows with USSTAB message 10.

Default: Value from PROLOG or previous SESSET command

logtime **Valid values:** 0- 60

logtime defines a time delay in seconds when using LOGTERM. During this time ***masc-ato*** waits with displaying the next screen.
With the command LOGTIME the time value in the dialog may be reset and redefined (see command LOGTIME on page 63).

Default: Value from PROLOG or previous SESSET command

logtry **Valid values:** 1 - 6000

logtry defines the number of attempts to establish a connection with the LOGTERM terminal. When using LOGTRY=1 the LOGTERM terminal is only used if available or if defined in VTAM. Otherwise the processing is continued without LOGTERM.

Default: Value from PROLOG or previous SESSET command

loop **Valid values:** 1 - 100000

This parameter serves as a LOOP-control. When exceeding this value the dialog is terminated.

Default: Value from PROLOG or previous SESSET command

mdtauto **Valid values:** YES, NO

When using MDTAUTO=YES , FILL-commands are generated automatically for MDT-display fields (eg. CICS BMS FSET fields). Fields which remain invisible to the user on the screen (Dark) and only serve for conversational prompting, may also be transmitted herewith.

When defining MDTAUTO=NO, only the fields that have been inserted or changed with FILL or MAPFLD, will be transmitted. This parameter sets the default for MAP MDTAUTO (see parameter MDTAUTO in the command MAP on page 65).

Default: Value from PROLOG or previous SESSET command

timeout **Valid values:** 0 - 60000

This parameter defines the max. time in seconds, while ***masc-ato*** waits for a reply. This parameter sets the default for MAP- and PENDING-TIMEOUT. The default value is sufficient for most applications.

When using TIMEOUT=1440 the timeout control is disabled. TIMEOUT=1440 should only be used for long running dialogs with HLLCALL's.

Default: 120

sessid **Valid values:** 1 character

This parameter specifies the session for which to use the parameters

Default: Value from PROLOG or previous SESSET command

support **Valid values:** NO, YES

This parameter is used for diagnostic help. With the use of SUPPORT=YES, detailed dialog evaluations are written to ATOLOG. When SUPPORT=YES is defined, you should use LOGTERM=YES for better performance.

Default: NO

VSE: When SUPPORT=YES is active, the JCL parameter //OPTION LOG,PARTDUMP has to be defined.

MVS: When SUPPORT=YES is active, the JCL parameter MSGLEVEL=(1,1) in the JOB card as well as a //SYSUDUMP DD statement has to be defined.

logmode **Valid values:** LOGMODE name according to VTAM definitions

The *logmode* parameter specifies which logmode should be applied for the virtual screen.

All screen models (model 2, 3, 4 or 5) are now supported.

Default: Value from PROLOG or previous SESSET command

logtmod **Valid values:** LOGMODE name according to VTAM definitions

This parameter defines the logmode for the LOGTERM. Usually this should be the same logmode as specified in the *logmode* parameter.

All screen models (model 2, 3, 4 or 5) are now supported.

Default: Value from PROLOG or previous SESSET command

Description:

The SESSSET command follows the SESSBEG and sets all parameters not corresponding with the ones set in the PROLOG. During a session, the parameters may be changed using SESSMOD.

Example:

```
SESSBEG SESSION=A
SESSSET SESSION=A,              C
```

```
          NETNAME=NETATO1,            C
          APPLID=DBDCCICS,            C
          LOGTERM=LOGON
        :
        SESSEND SESSION=A
        EPILOG
        END
```

Figure 74. Example SESSSSET

6.37. SLEEP

Syntax:

	Command	*Parameter*
	SLEEP	[**SEC**=*sec*]

sec **Valid values:** 1 - 3600.

Time value in seconds for the SLEEP command.

Default: 60

Description:

SLEEP is used to interrupt the running dialog. A SLEEP command does not read a new display screen, for this reason a new MAP, MAPEND sequence has to be defined.

Example:

```
        :
L1      MARK
        MAP
        FILL DATA='MENU'
        MAPEND
        SCAN DATA='MAIN-MENU',FOUND=L2
        PUTPRT DATA='WAIT FOR ENABLE CONDITION'
        SLEEP
        GOTO MARK=L1
        :
L2      MARK
        :
```

Figure 75. Example SLEEP

6.38. UEXIT

Syntax:

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

data **Valid values:** Max. 80 alphanumeric characters.

For control the data are transferred to the user exit ATOEXI, which returns the corresponding value in form of the user variables indicated by *to*.

Default: none

to **Valid values:** WORK1 - 9, FILL-Label or field declared by DCL.

to specifies the area into which the user exit places the requested data.

Default: WORK1

Description:

With UEXIT the user-exit is called explicitly. ATOEXI is used for the manipulation of fields with various possibilities. Further information on ATOEXI are to be found in appendix D on page 93.

Example:

```
        :
        UEXIT DATA='USER1',TO=WORK1
        PUTPRT FROM=WORK1
        FILL DATA='MENU'
        MOVE FROM=WORK1,TO=F1
        MAP
F1      FILL LIN=03,COL=04,DATA='XXXX'
        MAPEND
        :
```

Figure 76. Example UEXIT

7. EXECUTION OF ATO DIALOGS

This chapter deals with the procedure for the execution of ATO dialogs under VSE or MVS.

masc-ato commands can be embedded into any REXX programs. Jobs for the call of REXX procedures are provided on the SAMPLIB. There is no compilation or preprocessing necessary.

The ***masc-ato*** commands are defined as assembler macros and follow the assembler conventions. In order to execute a dialog, the macros first have to be converted, that is assembled and linked as a load module. It has to be considered that with the violation of the assembler conventions (see chapter "General Notation Rules" on page VIII), the corresponding error reports and messages will be displayed by the assembler program.

Basically, there are two methods to execute a ATO dialog:

- **Calling a dialog in the Link and Run-mode**
- **Calling a dialog in Go-mode**

Generally the first method is preferred and recommended. A comparison of the advantages and disadvantages of each method is discussed later in the chapter.

7.1. VSE Assembly with Link and Run

The VSE JCL displayed in figure 49 executes a ATO dialog, whereby the ***masc-ato*** commands are first assembled, linked as a load module and then executed.

```
* $$ JOB JNM=ATORUN
* $$ LST CLASS=L
* $$ LST CLASS=L,LST=01E
// JOB ATORUN
// OPTION LOG,PARTDUMP
// LIBDEF *,SEARCH=(userlib.ATO410),CATALOG=userlib.ATO410
// OPTION CATAL,NOXREF
   ACTION CLEAR
   PHASE DIALOG1,*
// EXEC ASSEMBLY
        PROLOG DIALOG=DIALOG1,...
        :
        EPILOG
        END
/*
// EXEC LNKEDT
// ASSIGN SYS001,01E
// ATORUN EXEC ATO,PARM='DIALOG=DIALOG1'
040           * SALES CODE   (INPUT FOR GETRDR)
02401         * ITEM NUMBER
/*
/&
* $$ EOJ
```

Figure 77. VSE Assembly with Link and Go

7.2. MVS Assembly with Link and Run

The MVS JCL displayed in figure 50 executes a ATO dialog whereby the ***masc-ato*** commands are first assembled, linked as a temporary load module and then executed.

```
//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           * SALES CODE        (INPUT FOR GETRDR)
02401         * ITEM NUMBER
/*
```

Figure 78. MVS Assembly with Link and Go

7.3. MVS Separate Dialog Generation and Execution

This method requires that you have assembled and linked the source of the ATO dialog into a user load library. The PARM field in the EXEC statement defines the ATO dialog to be executed with DIALOG=***dialogname***, where ***dialogname*** corresponds to the load module name in the ***user.loadlib***.

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

Figure 79. MVS ATO Dialog Generation

```
//job       JOB  ...
//ATOGO       EXEC PROC=ATOGO,DIALOG=dialogname,
//            REGION=2048K
//GO.STEPLIB DD DSN=user.loadlib,DISP=SHR
//GO.ATORDR DD *
040            * SALES CODE
02401          * ITEM NUMBER
/*
```

Figure 80. MVS Load Module Execution

7.4. Advantages and Disadvantages of the ATO Dialog Execution Methods

As mentioned in previous chapters, the two execution methods

- **Assembly with Link and Run-Mode**

  and

- **Calling the dialog in Go mode**

have various advantages and disadvantages. The first method with an **Assembly** followed by a **Link and Run** is to the preferred way to execute a ATO dialog. This method guarantees the consistency of all dialogs in a new environment after a ATO upgrade, without any re-assembling and re-linking of ATO dialogs.

In addition this method concentrates the whole dialog in only one job. This may be helpful in maintaining and adding new code to existing dialogs.

The second method consists of the calling of **assembled and linked load modules**. This method separates the source and the load modules of a dialog, which may be desirable or even required in installations with strict separation of authorities. Another advantage is, that at execution time, no assembly and link steps have to be run, which results in lower CPU load. These savings, however, are minimal on a MVS system.

+.

With a release change of ***masc-ato*** the dialogs should be assembled and newly linked.

We do recommend, that you use the Assembly with Link and Run method when executing a ATO dialog.

8. APPENDIX A - EXAMPLE DIALOGS

8.1. CICS Signon and Signoff

8.1.1. Call from 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
```

Figure 81. ATO Dialog Beispiel 1

Note

This example of a ***masc-ato*** dialog signs on to DBDCCICS succedded by a 'CESF LOGOFF'. If the sign-on is no successful, a return code of 8 is set.

The examples also shows how subroutines help to take frequently used code sequences out of the main program to make the program easier to read.

The values DATA='ATOUSR1' and DATA='ATO' could also be filled with DATA='#VAR1#' or. DATA='#VAR2#' respectively using the ATO User-Exit ATOEXI. For further details please look at the corresponding appendix and the chapter ATO_EXI.

This dialog can be found on the SAMPLB with the name RXAPPA1.

8.1.2. Compatibility Mode

```
       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 ? YES, SIGNOK                     *
************************************************
SIGNOK  MARK
LOGOFF  MARK
        MAP KEY=CLEAR
        MAPEND
```

```
MAP99   MAP LASTMAP=YES,DATA='CESF LOGOFF'
        MAPEND
        EPILOG
        END
```

Figure 82. ATO Dialog Example 1

Note

This example of a ***masc-ato*** dialog signs on to DBDCCICS succedded by a 'CESF LOGOFF'. If the sign-on is no successful, a return code of 8 is set.

The values DATA='ATOUSR1' and DATA='ATO' could also be filled with DATA='#VAR1#' or. DATA='#VAR2#' respectively using the ATO User-Exit ATOEXI.

8.2. CICS Command CEMT INQ TAS

8.2.1. Call from 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

...
```

Figure 83. ATO Dialog Example 2 from REXX

This example can be found on the SAMPLIB with the name RXAPPA2.

8.2.2. Compatiblity Mode

```
       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
```

Figure 84. ATO Dialog Example 2

8.3. Common CICS commands

8.3.1. Call from 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
/*
```

Figure 85. ATO Dialog Example 3 from REXX

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

8.3.2. Compatiblity Mode

```
       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
/*
```

Figure 86. ATO Dialog Example 3

9. APPENDIX B - MIGRATION TO VERSION 4.1

masc-ato 4.1.0 is source compatible to the previous version. All dialogs that are migrated have to be re-assembled and relinked.

Please note, that new the dialog name is specified directly at the program call, in the previous version it had to be passed as a parameter.

Also note that ***masc-ato*** 4.1.0 does not by itself translate any characters in uppercase but takes them as they appear on the screen. Please check if your dialog depends on this translation e.g. in SCAN commands. In this case, consider to turn on the uppercase translation of CICS for the virtual terminal.

10. APPENDIX C - USER EXIT ATOEXI

10.1. Introduction

The user exit ATOEXI is available for various requirements. ATOEXI is called at different positions of a ATO dialog implicitly by ***masc-ato*** or explicitly when using the command UEXIT or by FILL-variables #VARn#. Within the system the user exit ATOEXI is called at various points of the session handlings with #SYS#. The current APPLID from the TP monitor is tracked automatically by session passing.

After each alteration the user exit ATOEXI has to be newly generated , ie. according to figure 56 for VSE and figure 57 for MVS. Replace the library or data set names according to your ***masc-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
```

Figure 87. ATEXI Generation for 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
```

Figure 88. ATOEXI Generation for MVS

10.2. User Exit Variables #VARn#

The user variables #VAR1# - #VAR9# may be filled from REXX using ATO_EXI or in the compatibility mode in the FILL DATA parameter. With this the preset values in ATOEXI are set dependent on dialog names, APPLID, user ID etc. The dependency and transfer in user exit ATOEXI is determined by a table named TAB.

The transfer of variables may be used as entries for user ID, password, accounting information, company etc. Figure 58 shows part of the table TAB of ATOEXI for this purpose. This exit is supplied with the ***masc-ato*** software as an example. It may be tailored according to your requirements on your configuration, and generated newly.

For the call from REXX as well as for the compatibility mode the same module of ATOEXI is used, i.e., it has to be generated only once.

10.3. User Code in the Exit

The User Exit AOTEXI may also be used to insert your own code.

The place in the program for your user code is marked with the comment "`INSERT YOUR CODE HERE`" and "`END USER CODE`" respectively.

10.4. Using the User Exit

The following example explains the usage and functionality of the user exit ATOEXI.

```
         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
         :
```

Figure 89. User Exit Usage

With the definition of the user variables #VAR1# and #VAR2# in the DATA parameter of the FILL commands ATOEXI is called implicitly. At execution time the table TAB in ATOEXI is searched according to figure 59. In the above named example the variables #VAR1# and #VAR2# are replace by the values MILLER and ***masc-ato***. Any variables may generally be assigned with this table, if the corresponding conditions have been met. The corresponding replacement values have to be defined in the USER column. Per variable #VARn# (max. 9) and condition to be fulfilled, one line in the table TAB in ATOEXI has to be defined. The length of the line and the single fields within a line have to be followed exactly.

The contents of a line have been defined as follows:

```
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#
```

The field values may also be defined generically with '*' , whereby the asterisks (*) indicate the non-relevant positions.

The table TAB is run through from top to bottom (top-down) up to the first relevant entry. This is returned to the dialog in the FILL DATA of the corresponding #VARn# variable.

Incorrect:

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

Correct:

```
         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******'
         :
         :
```

Figure 90. User Exit ATOEXI (Part of table TAB)

11. INDEX