MiniWord

ToolKit Manual

DISCLAIMER

The information contained in this document is subject to change without notice.

Minisoft, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Minisoft, Inc. or its agents shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishings, performance, or use of this material.

LICENSE AGREEMENT

In return for payment of a onetime fee for this software product, the Customer receives from Minisoft, Inc. a license to use the product subject to the following terms and conditions:

- The product may be used on one computer system at a time: i.e., its use is not limited to a particular machine or user but to one machine at a time.
- The software may be copied for archive purposes, program error verification, or to replace defective media. All copies must bear copyright notices contained in the original copy.
- The software may not be installed on a network server for access by more than one personal computer without written permission from Minisoft, Inc.

Purchase of this license does not transfer any right, title, or interest in the software product to the Customer except as specifically set forth in the License Agreement, and Customer is on notice that the software product is protected under the copyright laws.

90-Day Limited Warranty

Minisoft, Inc. warrants that this product will execute its programming instructions when properly installed on a properly configured personal computer for which it is intended. Minisoft, Inc. does not warrant that the operation of the software will be uninterrupted or error free. In the event that this software product fails to execute its programming instructions, Customer's exclusive remedy shall be to return the product to Minisoft, Inc. to obtain replacement. Should Minisoft, Inc. be unable to replace the product within a reasonable amount of time, Customer shall be entitled to a refund of the purchase price upon the return of the product and all copies. Minisoft, Inc. warrants the medium upon which this product is recorded to be free from defects in materials and workmanship under normal use for a period of 90 days from the date of purchase. During the warranty period Minisoft, Inc. will replace media which prove to be defective. Customer's exclusive remedy for any media which proves to be defective shall be to return the media to Minisoft, Inc. for replacement.

ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE 90-DAY DURATION OF THIS WRITTEN WARRANTY. Some states or provinces do not allow limitations on how long an implied warranty lasts, so the above limitation or exclusion may not apply to you. This warranty gives you specific rights, and you may also have other rights which vary from state to state or province to province.

LIMITATION OF WARRANTY: Minisoft, Inc. makes no other warranty expressed or implied with respect to this product. Minisoft, Inc. specifically disclaims the implied warranty of merchantability and fitness for a particular purpose.

Table of Contents

Disclaimer 3

License Agreement 4

Minisoft Toolkit 7

Toolkit Libraries 21
- Files 21
- Functions 22

Create a Document - expanded table 41

Edit a Document - expanded table 42

Display a Document - expanded table 43

Copy a Document - expanded table 44

Convert a Document to an ASCII File - expanded table 46

Convert an ASCII File to Document - expanded table 48

Interactive Document Printing - expanded table 50

Background Document Printing - expanded table 53

Interactive List Processing - expanded table 56

Background List Processing - expanded table 59

Integrate Data and Text into a Document - expanded table 62

Concatenate Two Documents - expanded table 64

Delete a Document - expanded table 66

Set New File or Document Parameters - expanded table 67
Set List File or List Document - expanded table 72
Set Data Variable - expanded table 74
Set Text Variable - expanded table 75
Set Editor Parameters - expanded table 76
Exit - expanded table 79

MINISOFT TOOLKIT

The Minisoft ToolKit is a programmatic interface which allows application and system programmers to incorporate a comprehensive set of word processing features directly into their application programs. Since the ToolKit consists of a group of related programs, you will not need to recompile your software each time you receive a ToolKit upgrade. ToolKit's modular form should also help insure that future releases will be backwards compatible. The ToolKit offers the following features:

- creating documents
- editing documents
- displaying documents
- copying documents to MiniWord
- copying documents from MiniWord
- converting documents to ASCII files
- converting ASCII files to documents
- interactive document printing
- background document printing
- interactive list processing
- background list processing
- integrating data and text into documents
- setting new file and document parameters
- document assembly

The Minisoft ToolKit is a group of programs that other application programs call to perform word processing and document management tasks. It also allows you to convert documents to and from ASCII files (better known as EDITOR files on the HP e3000). The ToolKit gives you access to two different types of documents: standalone documents and documents owned by MiniWord. A standalone document resides on your system with the file name you supply as an output file name parameter. Documents owned by MiniWord

are assigned file names which are generated by the MiniWord system. You can reference these files through a 24 character document name and a 60 character folder name (group.account on MPE; the path on HP-UX). Documents owned by MiniWord are known to the MiniWord system; standalone documents are not. You should use standalone documents if your application program manages the documents or if you are only going to be using the document as a temporary file, i.e., created, printed, and deleted. If you need to access documents from both your application and from MiniWord, then use documents owned by MiniWord. Both types of documents can be used together to fit the needs of your application.

The ToolKit communicates with application programs through two IPC files on the HP e3000, or two named pipes on Unix machines. One file is a command file, the other a result file. Application programs must send the functions they want the ToolKit to execute via the command file and retrieve the results from the result file.

Before running the ToolKit, an application must create and open the two communication files. The communication file names need to have the form TKO##### and TKI#####. The five pound signs, #####, must be replaced with the calling application program's process ID number at the time the application program builds the actual files. The communication file names must follow this format. The I and the O, in TKI##### and TKO#####, refer to Input and Output as seen from the perspective of the calling application program. Think of TKO##### as the command file and TKI##### as the result file. On the HP e3000 the record size for TKO##### must be 256 bytes and the record size for TKI##### must be 64 bytes.

To give instructions to the ToolKit, the application must write to the command file. The ToolKit reads the command file, performs the corresponding function, and writes to the result file letting the application know whether or not the function executed successfully. The TKO##### and TKI##### files should always be written and read in pairs. The application program should write to the command file, TKO#####, then read from the result file, TKI#####, before proceeding with its next task.

To run the ToolKit, the calling application program must execute the program APPSRVR followed by the number 17 as a parameter. For example, on a HP e3000 running the MPE V operating system the application program must use the CREATEPROCESS intrinsic to run APPSRVR.PUB.MINISOFT with a PARM = 17. On HP e3000s running MPE XL the application program must use the CREATEPROCESS intrinsic to run APPSRVRN.PUB.MINISOFT with INFO="17". For HP-UX machines, the application

Table 1

General Command File Format			
Description	*Length*	*Offset*	*Values*
Function	1	0	79 - sample function
Reserved	7	1	
Input file type	1	8	33 - ASCII
Input file interface	1	9	32 - standalone file
Input file name	120	12	
Output file type	1	132	32 - document
Output file interface	1	133	32 - standalone file
Output file name	120	136	

should use the vfork() and exec() functions to create child processes and then run the program APPSRVR 17. The process of running the ToolKit can also be accomplished by calling the STARTTK function supplied in the ToolKit Library. Refer to the section on the ToolKit Library for more information about STARTTK.

When application programs send commands to the ToolKit via the file TKO#####, the ToolKit expects a block of 256 bytes that follows the general format shown below.

Note: For clarity, all values in this document are decimal numbers, base 10. Application programs should make conversion to base 2, base 8, or base 16, if necessary, when writing to the command file.

See *Table 1*.

The actual command file format is much more involved. Complete descriptions of all the different command formats are near the end of this document. The summarized command descriptions begin on the next page.

To send the command “Edit a Standalone Document” to the ToolKit, an application program would begin by writing the value 32 to position zero of a 256 byte buffer which mirrors the format of the command file TKO#####. Position zero of the command file is the function parameter. The ToolKit determines what function it will perform by reading this parameter. The input file type parameter at position eight determines the type of file the ToolKit will edit. In our example the application must write the value 32 in position eight to edit a document. The input file interface parameter specifies which type of document to edit: a standalone document of a document owned by MiniWord. The value for this parameter must be written at position nine.

In our example this would be 32. Starting with position 12 the application program should enter the input file name for the standalone document. The path or location should be included along with the file name. For example, on an HP e3000 the file could be called INFILE.PUB.MINI. On an HP9000 the same file would be called /usr/minisoft/infile. The 132nd position should contain the output file type, the 3rd position should contain the output file interface, and the 136th position should contain the output file name along with its location. In our example the output file type and the output file interface parameters are ignored, but the output file name parameter should be filled with a string of blank characters. All unused bytes should be set to zero. Once the application program has placed the appropriate values in the buffer, it must write the contents of the buffer to the command file to send it off to the ToolKit. Writing a command to the ToolKit can be accomplished with the WRITECF function supplied in the ToolKit Library. Refer to the section on the ToolKit Library for more information about WRITECF.

Not all the input parameters have to be filled in for every ToolKit function. The table below lists the TookKit functions and marks the necessary parameters with their decimal values, or with mnemonic symbols whose meanings are listed below.

Table 2 Legend:

B	A string of blank characters.
BTP	Blank characters to bring up an interactive screen that prompts for MWD. To include a centered title up to 64 characters long with your interactive screen, enter an exclamation point, then a blank space, then the title as your string, i.e..: “! Edit A Document”.
DM	Document owned by MiniWord.
DMP	Document owned by MiniWord with interactive prompt.
I	This field is ignored by the ToolKit.
MWD	MiniWord document name, folder and password.
SD	Standalone document.
SDF	Standalone document file name.

Table2

Function	Input file type	Inpute file interface	Input file name	Output file type	Output file interface	Output file name
32: create a document						
SD	I	I	B	32/34	32	SDF
DM	I	I	B	32/34	33	MWD
DMP	I	I	B	32/34	33	BTP
32: edit a document						
SD	32/34	32	SDF	I	I	B
DM	32/34	33	MWD	I	I	B
DMP	32.34	33	BTP	I	I	B
33: display a document						
SD	32	32	SDF	I	I	I
DM	32	33	MWD	I	I	I
DMP	32	33	BTP	I	I	I
34: copy a document						
SD -> SD	32	32	SDF	32	32	SDF
SD -> DM	32	32	SDF	32	33	MWD
SD -> DMP	32	32	SDF	32	33	BTP
DM -> SD	32	33	MWD	32	32	SDF
DM -> DM	32	33	MWD	32	33	MWD
DM -> DMP	32	33	MWD	32	33	BTP
DMP -> SD	32	33	BTP	32	32	SDF
DMP -> DM	32	33	BTP	32	33	MWD
DMP -> DMP	32	33	BTP	32	33	BTP

Table 3 Legend:

AFN	ASCII file name.
BTP	Blank characters to bring up an interactive screen to prompt for MWD; to include a centered title up to 64 characters long with your interactive screen, enter an exclamation point, a blank space, then the title as your string, i.e. "! Edit a Document".
DM	Document owned by MiniWord.
DMP	Document owned by MiniWord with interactive prompt.
MWD	MiniWord document name, folder and password.
SD	Standalone document.
SDF	Standalone document file name.
SPP	Supply printing parameters in the function call (the format of the print parameters is shown in the expanded table.
UPP	Use existing document parameters for printing.

Table3

Function	Input file type	Input file interface	Input file name	Output file type	Output file interface	Output file name
35: convert a document to an ASCII file						
SD -> ASCII	32	32	SDF	33	32	AFN
DM -> ASCII	32	33	MWD	33	32	AFN
DMP -> ASCII	32	33	BTP	33	32	AFN
35: convert an ASCII file to a document						
ASCII -> SD	33	32	AFN	32	32	SDF
ASCII -> DM	33	32	AFN	32	33	MWD
ASCII -> DMP	33	32	AFN	32	33	BTP
36: interactive document printing						
SD keep parms	32	32	SDF	32	0	UPP
SD set parms	32	32	SDF	32	1	SPP
DM keep parms	32	33	MWD	32	0	UPP
DM set parms	32	33	MWD	32	1	SPP
DMP keep parms	32	33	BTP	32	0	UPP
DMP set parms	32	33	BTP	32	1	SPP
37: background document printing						
SD keep parms	32	32	SDF	32	0	UPP
SD set parms	32	32	SDF	32	1	SPP
DM keep parms	32	33	MWD	32	0	UPP
DM set parms	32	33	MWD	32	1	SPP
DM keep parms	32	33	BTP	32	0	UPP
DMP set parms	32	33	BTP	32	1	SPP

Table 4 Legend:

AFN	ASCII file name.
BTP	Blank characters to bring up an interactive screen to prompt for MWD; to include a centered title up to 64 characters long with your interactive screen, enter an exclamation point, a blank space, then the title as your string, i.e. "! Edit a Document".
DM	Document owned by MiniWord.
DMP	Document owned by MiniWord with interactive prompt.
MWD	MiniWord document name, folder and password.
SD	Standalone document.
SDF	Standalone document file name.
SPP	Supply printing parameters in the function call (the format of the print parameters is shown in the expanded table.
UPP	Use existing document parameters for printing.

Table 4

Function	Input file type	Input file interface	Input file name	Output file type	Output file interface	Output file name
38: interactive list processing						
SD keep parms	32	32	SDF	32	0	UPP
SD set parms	32	32	SDF	32	1	SPP
DM keep parms	32	33	MWD	32	0	UPP
DM set parms	32	33	MWD	32	1	SPP
DMP keep parms	32	33	BTP	32	0	UPP
DMP set parms	32	33	BTP	32	1	SPP
39: background list processing						
SD keep parms	32	32	SDF	32	0	UPP
SD set parms	32	32	SDF	32	1	SPP
DM keep parms	32	33	MWD	32	0	UPP
DM set parms	32	33	MWD	32	1	SPP
DMP keep parms	32	33	BTP	32	0	UPP
DMP set parms	32	33	BTP	32	1	SPP
40: integrate data and text into a document						
data/text -> SD	32	32	SDF	I	I	I
data/text -> DM	32	33	MWD	I	I	I
data/text -> DMP	32	33	BTP	I	I	I
49: set list file or list document						
SD	32	32	SDF	I	I	I
DM	32	33	MWD	I	I	I
DMP	32	33	BTP	I	I	I
ASCII	33	32	AFN	I	I	I
127: exit	I	I	I	I	I	I

Table5

Function	Doc 1 file type	Doc 1 file interface	Doc 1 file name	Doc 2 file type	Doc 2 file interface	Doc 2 file name
41: concatenate two documents						
SD + SD	32/34	32	SDF	32/34	32	SDF
SD + DM	32/34	32	SDF	32/34	33	MWD
SD + DMP	32/34	32	SDF	32/34	33	BTP
DM + SD	32/34	33	MWD	32/34	32	SDF
DM + DM	32/34	33	MWD	32/34	33	MWD
DM + DMP	32/34	33	MWD	32/34	33	BTP
DMP + SD	32/34	33	BTP	32/34	32	SDF
DMP + DM	32/34	33	BTP	32/34	33	MWD
DMP + DMP	32/34	33	BTP	32/34	33	BTP

Table 5 Legend:

BTP	Blank characters to bring up an interactive screen to prompt for MWD; to include a centered title up to 64 characters long with your interactive screen, enter an exclamation point, a blank space, then the title as your string, i.e. "! Edit a Document".
DM	Document owned by MiniWord.
DMP	Document owned by MiniWord with interactive prompt.
MWD	MiniWord document name, folder and password.
SD	Standalone document.
SDF	Standalone document name.

Table 6

Function	File type	File interface	First offset	Last offset	Block length
48: setting new file or document parameters					
document	32	0	I	I	I
document	32	1	12	130	148
ASCII	33	0	I	I	I
ASCII	33	1	12	14	6

Table 6 Legend:

I	This field is ignored by the ToolKit.

Table 7

Function	Data variable name	Data variable value
50: set data variable		
	DVN	DVV

Table 7 Legend:

DVN	Up to 16 character data variable name.
DVV	Up to 128 character data variable value; the ToolKit can store 32 variables; a blank string resets all variables.

Table 8

Function	Data variable name	File type	File interface	File name
51: set text variable				
Text var =SD	DVN	32/34	32	SDF
Text var = DM	DVN	32/34	33	MWD
Test var = DMP	DVN	32/34	33	BTP

Table 8 Legend:

DVN	Up to 16 character data variable name.
BTP	Blank characters to bring up an interactive screen to prompt for MWD. To include a centered title up to 64 characters long with your interactive screen enter an exclamation point, a blank space, then the title as your string, i.e. "! Edit a Document".
DM	Document owned by MiniWord.
DMP	Document owned by MiniWord with interactive prompt.
MWD	MiniWord document name, folder and password.
SD	Standalone document.
SDF	Standalone document file name.

Table 9

Description	Length	Offset	Values
Return value	1	0	32 - success 33 - failure 34 - fatal failure
Error message	60	4	

See Table8

Once running, the ToolKit keeps processing all new instructions it receives from the command file. When the calling application program has no more instructions for the ToolKit to process, it should shut the ToolKit down before moving to its next task. The calling application program must send the exit command, 127, through the command file, read the result file, close the two communication files, and wait for the ToolKit and all the child processes created by the ToolKit to terminate. Shutting the ToolKit down can be accomplished with the STOPTK function supplied in the ToolKit Library. Refer to the section on the ToolKit Library for more information about STOPTK.

The ToolKit informs the calling application program as to whether or not the requested function executed successfully via the result file TKI#####. The file is 64 bytes long and follows the format below. Getting a result from the ToolKit can be accomplished with the READRF function supplied in the ToolKit Library. Refer to the section on the ToolKit Library for more information about READRF.

See Table 9.

The first byte of the result contains the return value. This value reports whether the ToolKit succeeded or failed to complete the requested task. A 32 in position zero indicates that the ToolKit successfully completed the task. A 33 in position zero signals that the ToolKit failed. The accompanying error message in position four of the result file explains why the ToolKit failed. In rare cases the ToolKit will return a 34, a fatal failure, in position zero of the result file. This error reveals that the ToolKit encountered an unexpected error from which it could not recover. The ToolKit terminates every time it encounters a fatal error. Application programs may want to read and display the error messages inside the application. All unused bytes in the result file are set to zero.

TOOLKIT LIBRARIES

FILES

Along with the ToolKit programs, the installation tape also contains source code and libraries for procedures that call and communicate with the ToolKit. The specific files are:

TKLIBSPL SPL source code for functions to start the ToolKit, stop the ToolKit, send commands to the ToolKit, and receive results from the ToolKit. Use these functions on MPE-V machines.

TKLIBC C source code for functions to start the ToolKit, stop the ToolKit, send commands to the ToolKit, and receive results from the ToolKit. Use these functions on MPE/iX machines.

TKLIBUSL MPE-V USL containing the compiled functions in TKLIBSPL.

TKLIBOBJ NMOBJ file containing the compiled functions in TKLIBC.

Programs using these functions will need PH capability.

FUNCTIONS

Function STARTTK

Syntax: I16 CA I16 I16 I16
return = STARTTK(*program, tk_pid, tk_cf, tk_rf)*

Use: STARTTK starts the ToolKit, creates and opens both the command IPC file and the result IPC file. STARTTK returns 0 if it is successful and -1 if it encounters an error.

Parameters:

program character array

The name of the ToolKit program terminated by a blank. Usually this is APPSRVRN.PUB.MINISOFT for MPE/iX, APPSRVR.PUB.MINISOFT for MPE-V non-PM version, and APPSRVRP.PUB.MINISOFT for MPE-V PM version. If the ToolKit programs are not installed in PUB.MINISOFT, change the group and account accordingly.

tk_pid 16-bit signed integer by reference.

The process id of the ToolKit is returned in this parameter. The STOPTK function uses this number.

tk_cf 16-bit signed integer by reference.

The file number of the command IPC file is returned in this parameter. The STOPTK and WRITECF functions use this number.

tk_rf 16-bit integer by reference.

The file number of the result IPC file is returned in this parameter. The STOPTK and READRF functions use this parameter.

Function STOPTK

Syntax: I16 I16V I16V I16V
return = STOPTK(*tk_pid, tk_cf, tk_rf)*

Use: STOPTK stops the ToolKit, closes and deletes both the command IPC file and the result IPC file. STOPTK returns 0 if it is successful and -1 if it encounters an error.

Parameters:

tk_pid 16-bit signed integer by value.

The process id of the ToolKit as returned by the STARTTK function.

tk_cf 16-bit signed integer by value.

The file number of the command IPC file as returned by the STARTTK function.

tk_rf 16-bit integer by value.

The file number of the result IPC file as returned by the STARTTK function.

Function WRITECF

Syntax: I16 I16V CA
return = WRITECF(*tk_cf, buffer)*

Use: WRITECF sends a command to the ToolKit. WRITECF returns 0 if it is successful and -1 if it encounters an error.

Parameters:

tk_cf 16-bit signed integer by value.

The file number of the command IPC file as returned by the STARTTK function.

buffer Character array.

The 256 character array containing the command and the parameters to be sent to the ToolKit.

Function READRF

Syntax: I16 I16V CA
return = READRF(*tk_rf, buffer)*

Use: READRF receives the result of the last command sent to the ToolKit. READRF returns 0 if it is successful and -1 if it encounters an error.

Parameters:

tk_rf 16-bit signed integer by value.

The file number of the result IPC file as returned by the STARTTK function.

buffer Character array.

The result of the last command sent to the ToolKit is returned in this parameter. The size of the array must be at least 64 characters.

Definition of symbols:

I16 16-bit signed integer.

CA Character array.

If a symbol is followed by a ‘V’, the parameter is passed by value, otherwise the parameter is passed by reference.

Notes:

The functions must be declared as external and of the proper type in the source code of the calling application. Failure to do this may cause the calling application to assume the function type is different from its actual type. This can cause stack overflows and underflows.

Make sure the application passes the correct number and type of parameters. The address of the actual parameter should be passed for parameters passed by reference. The value of the actual parameter should be passed by value. Mistakes in either the number, type, or passing type of parameters can cause very unpredictable results.

These functions can be used from source code, USL (MPE-V), RL (MPE-V or MPE/iX), SL (MPE-V), or XL (MPE/iX). Use the Segmenter on MPE-V to create RLs and SLs. Use the Link Editor/XL on MPE/iX to create RLs and XLs.

The following is a listing of the file TKLIBSPL:

```
$CONTROL USLINIT,SUBPROGRAM

BEGIN

<< Example procedures to start, stop and communicate with the ToolKit >>
<<     for MPE-V                                          >>

<< receive results from the ToolKit >>
INTEGER PROCEDURE READRF(TK'RF, BUF);
  VALUE TK'RF;
  INTEGER TK'RF;
  BYTE ARRAY BUF;
BEGIN
  INTRINSIC FREAD;

  FREAD(TK'RF, BUF, -64);
  IF <> THEN BEGIN
    READRF := -1;
  END ELSE BEGIN
    READRF := 0;
  END;
END;

<< send a command to the ToolKit >>
INTEGER PROCEDURE WRITECF(TK'CF, BUF);
  VALUE TK'CF;
  INTEGER TK'CF;
  BYTE ARRAY BUF;
BEGIN
  INTRINSIC FWRITE;

  FWRITE(TK'CF, BUF, -256, 0);
  IF <> THEN BEGIN
    WRITECF := -1;
  END ELSE BEGIN
```

```
    WRITECF := 0;
  END;
END;

<< this procedure creates the two IPC files needed to communicate with >>
<< the ToolKit, and starts the ToolKit program >>
INTEGER PROCEDURE STARTTK(PROG, TK'PID, TK'CF, TK'RF);
  BYTE ARRAY PROG;
  INTEGER TK'PID, TK'CF, TK'RF);
BEGIN
  INTRINSIC PROCINFO, ASCII, FOPEN, FCLOSE, FCHECK,
    CREATEPROCESS, ACTIVATE;
  BYTE ARRAY BUF(0:5), IP'NM(0:8), OP'NM(0:8);
  INTEGER ARRAY ITEMNUM(0:2);
  LOGICAL ARRAY ITEMS(0:2);
  INTEGER I,J, K, ERR, ERR1, ERR2;
  INTEGER IT1, IT2, PIN;

<< get this processes process id >>
  IT1 := 1;
  IT2 := 0;
  PROCINFO(ERR1, ERR2, 0, IT1, PIN, IT2, BUF);
  IF <> THEN BEGIN
    STARTTK := -1;
    RETURN;
  END;

<< IPC file names are TKO##### and TKI#####, where TKO##### is the command >>
<< output IPC file and TKI##### is the status return IPC file, and where ##### >>
<< is this processes process id, right justified and zero filled. >>
  I := ASCII(PIN, 10, BUF);
  J := 0;
  FOR K := 3 UNTIL 7 DO BEGIN
    IF I < (8 - K) THEN BEGIN
      IP'NM(K) := OP'NM(K) := "0";
    END ELSE BEGIN
```

```
    IP'NM(K) := OP'NM(K) := BUF(J);
    J := J + 1;
   END;
  END;
  IP'NM(0) := OP'NM(0) := "T";
  IP'NM(1) := OP'NM(1) := "K";
  IP'NM(2) := "I";
  OP'NM(2) := "O";
  IP'NM(8) := OP'NM(8) := " ";

<< check to see if the status return IPC file already exists.  If it >>
<< exists, purge it.  >>
<< foptions: >>
<<  Domain - 10, old temp file >>
<<  ASCII/Binary - 0, Binary >>
<<  Default file Desig. - 000 >>
<<  Record Format - 01, variable length records >>
<<  Disallow File Equation - 1, Disallow :FILE >>
<< Aoptions:  >>
<<  Access type - 0000, Read access >>
<<  Exclusive - 01, Exclusive access >>
<<  Multi-Access - 01, Intra-job multi-access >>

  I := FOPEN(IP'NM, %(16)0442, %(16)0240);
  IF <> THEN BEGIN
    FCLOSE(I, 4, 0);
  END ELSE BEGIN
    FCHECK(I, K);
    IF K = 0 THEN BEGIN
      FCLOSE(I, 4, 0);
    END;
  END;
<< create the status return IPC file, close and save it, then reopen it >>
<< foptions: >>
<<  Domain - 00, new file >>
<<  ASCII/Binary - 0, Binary >>
<<  Default File Desig. - 000 >>
```

```
<<  Record Format - 10, variable length records >>
<<  Disallow File Equation - 1, Disallow :FILE >>
<<  File Type - IPC file >>
<< Aoptions:  >>
<<  N/A  >>

 I := FOPEN(IP'NM, %(16)3440,,32,,,,1,,2D,1,1);
 FCLOSE(I, 2, 0);
<< foptions:  >>
<<  Domain - 10, old temp file >>
<<  ASCII/binary - 0, binary >>
<<  Default File Desig. - 000 >>
<<  Record Format - 01, variable length records >>
<<  Disallow File Equation - 1, Disallow :FILE >>
<< Aoptions:  >>
<<  Access Type - 0000, Read access  >>
<<  Exclusive - 01, Exclusive access, >>
<<  Multi-Access - 01, Intra-job multi-access >>

 I := FOPEN(IP'NM, %(16)0442, %(16)0240);
 TK'RF := I;

<< check to see if the command IPC file already exists.  If it >>
<< exists, purge it. >>
<< foptions: >>
<<  Domain - 10, old temp file >>
<<  ASCII/Binary - 0, binary  >>
<<  Default File Desig. - 000 >>
<<  Record format - 01, variable length records >>
<<  Disallow File Equations - 1, Disallow :FILE >>
<< Aoptions:  >>
<<  Access Type - 0001, Write access >>
<<  Exclusive - 01, Exclusive access >>
<<  Multi-Access - 01, Intra-job multi-access >>

 I := FOPEN(OP'NM, %(16)0442, %(16)0241);
 IF <> THEN BEGIN
```

```
    FCLOSE(I, 4, 0);
  END ELSE BEGIN
    FCHECK(I, J);
    IF K = 0 THEN BEGIN
      FCLOSE(I, 4, 0);
    END;
  END;
<< create the command IPC file, close it and save it, then reopen it >>
<< foptions:  >>
<<  Domain - 00, new file  >>
<<  ASCII/Binary - 0, Binary >>
<<  Default File Desig. - 000  >>
<<  Record Format - 01, variable length records >>
<<  Disallow File Equations - 1, Disallow :FILE >>
<<  File Type - IPC file  >>
<< Aoptions:  >>
<<  N/A  >>

  I := FOPEN(OP'NM, %(16)3440,,128,,,,1,,2D,1,1);
  FCLOSE(I, 2, 0);
<< foptions:  >>
<<  Domain - 10, old temp file  >>
<<  ASCII/Binary - 0, Binary  >>
<<  Default File Desig. - 000  >>
<<  Record Format - 01, variable length records >>
<<  Disallow File Equation - 1, Disallow :FILE  >>
<< Aoptions:  >>
<<  Access Type - 0001, Write access >>
<<  Exclusive - 01, Exclusive access,  >>
<<  Multi-Access - 01, Intra-job multi-access  >>

  I := FOPEN(OP'NM, %(16)0442, %(16)0241);
  TK'CF := I;

<< set parameters for CREATEPROCESS intrinsic  >>
<< load option flags - Bit 15 = 1; >>
  ITEMNUM(0) := 3;
  ITEMS(0) := 1;
```

```
<< set PARM value to 17  >>
  ITEMNUM(1) := 2;
  ITEMS(1) := 17;
  ITEMNUM(2) := 0;
  ITEMS(2) := 0;

<< create and activate the ToolKit  >>
  CREATEPROCESS(ERR, TK'PID, PROG, ITEMNUM, ITEMS);
  IF <> THEN BEGIN
    STARTTK := -1;
    RETURN;
  END;
  ACTIVATE(TK'PID);
  IF <> THEN BEGIN
    STARTTK := -1;
    RETURN;
  END;
  STARTTK := 0;
END;

<< this procedure stops the ToolKit to terminate, then  >>
<< closes and purges the IPC files.  >>
INTEGER PROCEDURE STOPTK(TK'PID, TK'CF, TK'RF);
  VALUE TK'PID, TK'CF, TK'RF;
  INTEGER TK'PID, TK'CF, TK'RF;
BEGIN
  INTRINSIC FCLOSE, GETPROCINFO, PAUSE;
  REAL T;
  DOUBLE STAT;
  INTEGER I;
  BYTE ARRAY BUF(0:255);
  LOGICAL EXIT;

<< send exit command to ToolKit  >>
  MOVE BUF(1) := 255(" ");
  BUF(0) := 127;
  IF WRITECF(TK'CF, BUF) = -1 THEN BEGIN
    STOPTK := -1;
    RETURN;
```

```
  END;
  IF READRF(TK'RF, BUF) = -1 THEN BEGIN
    STOPTK := -1;
    RETURN;
  END;

<< close and purge IPC files  >>
  FCLOSE(TK'RF, 4, 0);
  IF <> THEN BEGIN
    STOPTK := -1;
    RETURN;
  END;
  FCLOSE(TK'CF, 4, 0);
  IF <> THEN BEGIN
    STOPTK := -1;
    RETURN;
  END;
<< wait for ToolKit to terminate  >>
  T:= 1.0;
  EXIT := FALSE;
  DO BEGIN
    STAT := GETPROCINFO(TK'PID);
    IF <> THEN BEGIN
      EXIT := TRUE;
    END ELSE BEGIN
      PAUSE(T);
    END;
   END UNTIL EXIT = TRUE;
   STOPTK := 0;
 END;

END.
```

The following is a listing of the file TKLIBC:

```
/* Example procedures to start, stop and communicate with the ToolKit */
/*   FOR MPE/iX                                                        */

#include <mpe.h>

#pragma intrinsic ASCII
#pragma intrinsic PROCINFO
#pragma intrinsic FOPEN
#pragma intrinsic FCLOSE
#pragma intrinsic FREAD
#pragma intrinsic FWRITE
#pragma intrinsic FCHECK
#pragma intrinsic PAUSE
#pragma intrinsic CREATEPROCESS
#pragma intrinsic ACTIVATE
#pragma intrinsic GETPROCINFO

/* this procedure creates the two IPC files needed to communicate with
   the ToolKit, and starts the ToolKit program */
short int STARTTK(prog, tk_pid, tk_cf, tk_rf)
  char *prog;
  short int *tk_pid, *tk_cf, *tk_rf;
{
  short int i, j, k, pin, err1, err2;
  int itemnum[4], items[4], err;
  char buf[6], ip_nm[9], op_nm[9];

/* get this processes process id */
  PROCINFO(&err1, &err2, 0, 1, &pin);
  if (ccode() == CCL) {
    return(-1);
  }

/* IPC file names are TKI##### and TKO######, where TKO##### is the command
   output IPC file and TKI##### is the status return IPC file, and where ######
```

```
   is this processes process id, right justified and zero filled. */
  i = ASCII(pin, 10, buf);
  for (k = 3, j = 0; k < 8; ++k) {
    ip_nm[k] = op_nm[k] = (i < 8 - k) ? '0' :buf[j++];
  }
  ip_nm[0] = op_nm[0] = 'T';
  ip_nm[1] = op_nm[1] = 'K';
  ip_nm[2] = 'I';
  op_nm[2] = 'O';
  ip_nm[8] = op_nm[8] = ' ';

/* check to see if the status return IPC file already exists.  If it /*
exists, purge it. */
/* foptions:
     Domain - 10, old temp file
     ASCII/Binary - 0, Binary
     Default File Desig. - 000
     Record Format - 01, variable length records
     Disallow File Equations - 1, Disallow :FILE
   Aoptions:
     Access Type - 0000, Read access
     Exclusive - 01, Exclusive access
     Multi-Access - 01, Intra-job multi-access
*/
  i = FOPEN(ip_nm, 0x0442, 0x0240);
  if (ccode() == CCE) {
    FCLOSE(i, 4, 0);
  } else {
    FCHECK(i, &k);
    if (k == 0) {
      FCLOSE(i, 4, 0);
    }
  }
/* create the status return IPC file, close and save it, then reopen it */
/* foptions:
     Domain - 00, new file
     ASCII/Binary - 0, Binary
```

```
    Default File Desig. - 000
    Record Format - 01, variable length records
    Disallow File Equations - 1,  Disallow :FILE
    File Type - IPC file
   Aoptions:
    N/A
*/
  i = FOPEN(ip_nm, 0x3440,,32,,,,1,,2L,1,1);
  FCLOSE(i, 2, 0);
/* foptions:
    Domain - 10, old temp file
    ASCII/Binary - 0, Binary
    Default File Desig. - 000
    Record Format - 01, variable length records
    Disallow File Equation - 1, Disallow :FILE
   Aoptions:
    Access Type - 0000, Read Access
    Exclusive - 01, Exclusive access,
    Multi-Access - 01, Intra-job multi-access
*/
  i = FOPEN(ip_nm, 0x0442, 0x0240);
  *tk_rf = i;

/* check to see if the command IPC file already exists.  If it
   exists, purge it. */
/* foptions:
    Domain - 10, old temp file
    ASCII/Binary - 0, Binary
    Default File Desig. - 000
    Record format - 01, variable length records
    Disallow File Equation - 1, Disallow :FILE
   Aoptions:
    Access Type - 0001, Write access
    Exclusive - 01, Exclusive access
    Multi-Access - 01, Intra-job multi-access
*/
  i = FOPEN(op_nm, 0x0442, 0x0241);
```

```
  if (ccode() == CCE) {
    FCLOSE(i, 4, 0);
  } else {
    FCHECK(i, &k);
    if (k == 0) {
      FCLOSE(i, 4, 0);
    }
  }
/* create the command IPC file, close and save it, then reopen it */
/* foptions:
    Domain - 00, new file
    ASCII/Binary - 0, Binary
    Default File Desig. - 000
    Record Format - 01, variable length records
    Disallow File Equations - 1, Disallow :FILE
    File Type - IPC file
   Aoptions:
    N/A
*/
  i = FOPEN(iop_nm, 0x3440 ,, 128 ,,,, 1 ,, 2L, 1, 1);
  FCLOSE(i, 2, 0);
/* foptions:
    Domain - 10, old temp file
    ASCII/Binary - 0, Binary
    Default File Desig. - 000
    Record Format - 01, variable length records
    Disallow File Equation - 1, Disallow :FILE
   Aoptions:
    Access Type - 0001, Write access
    Exclusive - 01, Exclusive access
    Multi-Access - 01, Intra-job multi-access
*/
  i = FOPEN(op_nm, 0x0442, 0x0241);
  *tk_cf = i;

/* set parameters for CREATEPROCESS intrinsic */
/* load option flags - Bit 15 = 1 */
```

```
  itemnum[0] = 3;
  items[0] = 0x01;
/* set INFO string to “17” */
  itemnum[1] = 11;
  items[1] = (int) “17”;
  itemnum[2] = 12;
  items[2] = 2;
  itemnum[3] = 0;
  items[3] = 0;
/* create and activate the ToolKit */
  CREATEPROCESS(&err, tk_pid, prog, itemnum, items);
  if (ccode() != CCE) {
    return(-1);
  }
  ACTIVATE(*tk_pid);
  if (ccode() != CCE) {
    return(-1);
  }
  return(0);
}

/* this procedure stops the ToolKit to terminate, then
   closes and purges the IPC files. */
short int STOPTK(tk_pid, tk_cf, tk_rf)
 short int tk_pid, tk_cf, tk_rf;
{
 float t;
 long stat;
 short int i, WRITECF(), READRF();
 char buf[256];

/* send exit command to ToolKit */
 for (i = 0; i < 256; ++i) {
   buf [i] = ‘ ‘;
 }
 buf[0] = 0x7F;
 if (WRITECF(tk_cf, buf) == -1) {
```

```
  return(-1);
 }

 if (READRF(tk_rf, buf) == -1) {
  return(-1);
 }
/* close and purge IPC files */
  FCLOSE(tk_rf, 4, 0);
  if (ccode() != CCE) {
   return(-1);
  }
  FCLOSE(tk_cf, 4, 0);
  if (ccode () != CCE) {
   return(-1);
  }
/* wait for ToolKit to terminate */
  t = 1;
  do {
   stat = GETPROCINFO(tk_pid);
   if (ccode() != CCE) break;
   PAUSE(&t);
  } while(1);
  return(0);
}

/* receive result from the ToolKit */
short int READRF(tk_rf, buf)
  short int tk_rf;
  char *buf;
{
  FREAD(tk_rf, (short unsigned *) buf, -64);
  if (ccode() != CCE) {
    return(-1);
  }
  return(0);
}
```

```
/* send a command to the ToolKit */
short int WRITECF(tk_cf, buf)
 short int tk_cf;
 char *buf;
{
 FWRITE(tk_cf, (short unsigned *) buf, -256, 0);
 if (ccode() != CCE {
  return(-1)
 }
 return(0);
}
```

CREATE A DOCUMENT - EXPANDED TABLE

This function creates a new document or glossary by using the MiniWord editor. It enables you to create either a standalone document or a document owned by MiniWord. New documents receive their formatting parameters from the defaults set in MiniWord or from the parameters set with ToolKit function 48. The output file name specifies both the name and location of the document. See *Table 10*.

Table 10

Description	Length	Offset	Values
Function	1	0	32 - create/edit
Reserved	7	1	
Input file type	1	8	ignored
Input file interface	1	9	iginored
Input file name	120	12	blank string
Output file type	1	132	32 - document 34 - glossary
Output file interface	1	133	32 - standalone file 33 - document owned by MiniWord
For output file interface = 32:			
Output file name	120	136	
For output file interface = 33:			
Document name supplied?	1	136	32 - yes 33 - no
For document name supplied = 32:	24	138	
Document name	60	162	
Folder name	8	222	
Password			
For document name supplied = 33:			
Optional menu title	64	138	

EDIT A DOCUMENT - EXPANDED TABLE

This function calls the MiniWord editor to edit the document or glossary specified by the input file name. You may edit a standalone document or document owned by MiniWord. See *Table 11*.

Table 11

Description	Length	Offset	Values
Function	1	0	32 - create/edit
Reserved	7	1	
Input file type	1	8	32 - document 34 - glossary
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplies = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Output file type	1	132	ignored
Output file interface	1	133	ignored
Output file name	120	136	blank string

DISPLAY A DOCUMENT - EXPANDED TABLE

This function displays the contents of the document specified by the input file name onto the CRT screen. You can display either a standalone document or a document owned by MiniWord. See *Table 12.*

Table 12

Description	Length	Offset	Values
Function	1	0	33 - display
Reserved	7	1	
Input file type	1	8	32 - document
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Output file type	1	132	ignored
Output file interface	1	133	ignored
Output file name	120	136	ignored

COPY A DOCUMENT - EXPANDED TABLE

This function copies the source document, specified by the input file name, to the destination document, specified by the output file name. Both the source document and the destination document may be either standalone documents or documents owned by MiniWord.

If the destination document already exists, the copy function overwrites it with the source document. However, if you are copying a document owned by MiniWord and are using the prompt option, you will be given a choice to either overwrite the destination document or cancel the copy operation.

MAIN USES:

- To copy a document owned by MiniWord to a standalone document, or to copy a standalone document to a document owned by MiniWord.
- To make many copies of a single master document. Many applications generate documents that vary only slightly from one document, called the Master. Use the copy function to create working copies which can be modified without affecting the master.
- For document archiving and transfer procedures.

See *Table 13*.

Table 13

Description	Length	Offset	Values
Function	1	0	34 - copy
Reserved	7	1	
Input file type*	1	8	32 - document 34 - glossary
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Output file type*	1	132	32 - document 34 - glossary
Output file interface	1	133	32 - standalone file 33 - document owned by MiniWord
For output file interface = 32:			
Output file name	120	136	
For output file interface = 33:			
Document name supplied?	1	136	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	138	
Folder name	60	162	
Password	8	222	
For document name supplied = 33:			
Optional menu title	64	138	

CONVERT A DOCUMENT TO AN ASCII FILE - EXPANDED TABLE

This function converts a document, specified by the input file name, to an ASCII file, specified by the output file name. The document may be a standalone document or a document owned by MiniWord. Set the ASCII file record size and file size with ToolKit function 48 (on the HPe3000 only, you may use the defaults for the record size and file size which are 200 and 10000 respectively). Set the record size to the largest right margin used in your document and set the file size to the maximum number of lines in your document. Once the document is converted to an ASCII file, the ASCII file is truncated at its EOF to remove any unused space.

MAIN USES:

- To convert a document to an ASCII file to merge it into a report generated by an application program.
- To convert a document to an ASCII file to pass it to an application that requires ASCII text as input, i.e. TELEX, FAX or typesetting applications.
- To convert a document to an ASCII file to store it in a database. Many applications only allow ASCII text to be stored in their data bases. An application program can use the ToolKit to create text with MiniWord and convert it to an ASCII file so that it can be stored in the database in the proper format.

See *Table 14*.

Table 14

Description	Length	Offset	Values
Function	1	0	35 - convert
Reserved	7	1	
Input file type	1	8	32 - document
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Output file type	1	132	33 - ASCII
Output file interface	1	133	32 - ASCII file
Output file name	120	136	ASCII file name

CONVERT AN ASCII FILE TO DOCUMENT

- EXPANDED TABLE

This function converts an ASCII file, specified by the input file name, to a document, specified by the output file name. The document may be a standalone document or one owned by MiniWord. If the document already exists, the conversion function overwrites it with the converted copy of the ASCII file. However, if you are converting to a document owned by MiniWord and are using the prompt option, you will be given a choice to either overwrite the MiniWord or cancel the conversion operation.

The new document receives its formatting parameters from the defaults set in MiniWord or from the parameters set with ToolKit function 48. In order to format the document correctly the left and right margin parameters must match the left and right margins of the ASCII file. This insures proper paragraph formatting and elimination of leading blanks.

MAIN USES:

- To convert ASCII output from an application into document format to take advantage of MiniWord's word processing features.

See *Table 15*.

Table 15

Description	Length	Offset	Values
Function	1	0	35 - convert
Reserved	7	1	
Input file type	1	8	33 - ASCII
Input file interface	1	9	32 - ASCII file
Input file name	120	12	ASCII file name
Output file type	1	132	32 - document
Output file interface	1	133	32 - standalone file 33 - document owned by MiniWord
For output file interface = 32:			
Output file name	120	136	
For output file interface = 33:			
Document name supplied?	1	136	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	138	
Folder name	60	162	
Password	8	222	
For document name supplied = 33:			
Optional menu title	64	138	

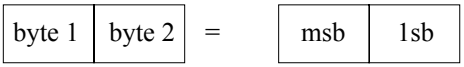
INTERACTIVE DOCUMENT PRINTING -

EXPANDED TABLE

This function prints the document specified by the input file name. The document may be a standalone document or a document owned by MiniWord. The printing function uses the MiniWord print menu which gives users interactive control over the printing process.

The print menu can gather printing parameters either from the document or from the function request itself (see following table). If the function request supplies one parameter, it must supply all other parameters as well.

The format for all parameters which represent numbers and are two bytes long (two-byte numerics) must have the most significant bits of the number in byte one and the least significant bits of the number in byte two, as illustrated in the diagram below:



An example of a two-byte numeric is the “Start printing with page #” parameter.

See *Table 16* and *Table 17*.

Table 16

Description	Length	Offset	Values
Function	1	0	36 - interactive print
Reserved	7	1	
Input file type	1	8	32 - document
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Use original defaults?	1	135	0 - yes 1 - no
Printer name	8	136	printer defined to Manager
Print mode	1	144	0 - continuous 1 - single sheet 2 - pause 1st page 3 - no form feeds 4 - twin sheet feeder
Copies	1	145	1 to 99
Left margin	1	146	0 to 99
Line spacing	1	147	0 - single spacing 1 - double spacing
Font	1	148	0 - draft 1 - letter quality

Table 17

Description	Length	Offset	Values
Global initial sequences #	1	149	0 - initial sequence 1 1 - initial sequence 2 2 - initial sequence 3 3 - initial sequence 4 4 - initial sequence 5
Global terminating seq. #	1	150	0 - terminating seq. 1 1 - terminating seq. 2 2 - terminating seq. 3 3 - terminating seq. 4 4 - terminating seq. 5
Starting printing with page #	2	152	0 to 9999
End printing after page #	2	154	0 to 9999
Start page number with	2	156	0 to 9999
Start headings on page #	2	158	0 to 9999
Start footings on page #	2	160	0 to 9999

BACKGROUND DOCUMENT PRINTING -

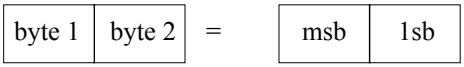
EXPANDED TABLE

This function prints the document specified by the input file name. The document may be a standalone document or a document owned by MiniWord.

The background document printing function is almost identical to the interactive document printing function except that it does not use the MiniWord print menu. All printing is done behind the scenes.

The print menu can gather printing parameters either from the document or from the function request itself (see the table below). If the function request supplies one parameter, it must supply all other parameters as well.

The format for all parameters which represent numbers and are two bytes long (two-byte numerics) must have the most significant bits of the number in byte one and the least significant bits of the number in byte two, as illustrated in the diagram below:



An example of a two-byte numeric is the “Start printing with page #” parameter.

See *Table 18* and *Table 19*.

Table 18

Description	Length	Offset	Values
Function	1	0	37 - background print
Reserved	7	1	
Input file type	1	8	32 - document
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Use original defaults?	1	135	0 - yes 1 - no
Printer name	8	136	0 - continuous
Print mode	1	144	1 - single sheet 2 - pause 1st page 3 - no form feeds 4 - twin sheet feeder
Copies	1	145	1 to 99
Left margin	1	146	0 to 99
Line spacing	1	147	0 - single spacing 1 - double spacing

Table 19

Description	Length	Offset	Values
Font	1	148	0 - draft 1 - letter quality 2 - proportional
Global initial sequence #	1	149	0 - initial sequence 1 1 - initial sequence 2 2 - initial sequence 3 3 - initial sequence 4 4 - initial sequence 5
Global terminating sequence #	1	150	0 - terminating seq. 1 1 - terminating seq. 2 2 - terminating seq. 3 3 - terminating seq. 4 4 - terminating seq. 5
Start printing with page #	2	152	0 to 9999
End printing after page #	2	154	0 to 9999
Start page number with	2	156	0 to 9999
Start headings on page #	2	158	0 to 9999
Start footings on page #	2	160	0 to 9999

INTERACTIVE LIST PROCESSING - EXPANDED

TABLE

This function merges data from a list with a form document specified by the input file name. The resulting output is sent to a printer. The form document may be a standalone document or a document owned by MiniWord.

The interactive list processing function is almost identical to the interactive document printing function except that the list of data must be defined with ToolKit function 49 before the function can be used.

The print menu, which lets you control list processing, can gather printing parameters either from the form document or from the function request itself (see the table below). If the function request supplies one parameter, it must supply all other parameters as well.

The format for all parameters which represent numbers and are two bytes long (two-byte numerics) must have the most significant bits of the number in byte one and the least significant bits of the number in byte two, as illustrated:

byte 1	byte 2	=	msb	lsb

An example of a two-byte numeric is the “Start printing with page #” parameter.

See *Table 20* and *Table 21*.

Table 20

Description	Length	Offset	Values
Function	1	0	38 - interactive list processing
Reserved	7	1	
Input file type	1	8	32 - document
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Use original defaults?	1	135	0 - yes 1 - no
Printer name	8	136	0 - continuous
Print mode	1	144	1 - single sheet 2 - pause 1st page 3 - no form feeds 4 - twin sheet feeder

Table 21

Description	Length	Offset	Values
Copies	1	145	1 to 99
Left margin	1	146	0 to 99
Line spacing	1	147	0 - single spacing 1 - double spacing
Font	1	148	0 - draft 1 - letter quality 2 - proportional
Global initial sequence #	1	149	0 - initial sequence 1 1 - initial sequence 2 2 - initial sequence 3 3 - initial sequence 4 4 - initial sequence 5
Global terminating seq. #	1	150	0 - terminating seq. 1 1 - terminating seq. 2 2 - terminating seq. 3 3 - terminating seq. 4 4 - terminating seq. 5
Start printing with page #	2	152	0 to 9999
End printing after page #	2	154	0 to 9999
Start page number with	2	156	0 to 9999
Start headings on page #	2	158	0 to 9999
Start footings on page #	2	160	0 to 9999

BACKGROUND LIST PROCESSING -

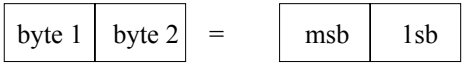
EXPANDED TABLE

This function merges data from a list with a form document specified by the input file name without using the interactive menu. The resulting output is sent to a printer. The form document may be a standalone document or a document owned by MiniWord.

The background list processing function is almost identical to the interactive list processing function except that it does not use the print menu. As with interactive list processing the list of data must be defined with ToolKit function 49 before the function can be used.

Printing parameters can be taken from either the form document or from the function request itself (see the table below). If the function request supplies one parameter, it must supply all other parameters as well.

The format for all parameters which represent numbers and are two bytes long (two-byte numerics) must have the most significant bits of the number in byte one and the least significant bits of the number in byte two. As illustrated:



An example of a two-byte numeric is the “Start printing with page #” parameter.

See *Table 22* and *Table 23*.

Table 22

Description	Length	Offset	Values
Function	1	0	39 - background list processing
Reserved	7	1	
Input file type	1	8	32 - document
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Use original defaults?	1	135	0 - yes 1 - no
Printer name	8	136	
Print mode	1	144	0 - continuous 1 - single sheet 2 - pause 1st page 3 - no form feeds 4 - twin sheet feeder

Table 23

Description	Length	Offset	Values
Copies	1	145	1 to 99
Left margin	1	146	0 to 99
Line spacing	1	147	0 - single spacing 1 - double spacing
Font	1	148	0 - draft 1 - letter quality 2 - proportional
Global initial sequence #	1	149	0 - initial sequence 1 1 - initial sequence 2 2 - initial sequence 3 3 - initial sequence 4 4 - initial sequence 5
Global terminating seq. #	1	150	0 - terminating seq. 1 1 - terminating seq. 2 2 - terminating seq. 3 3 - terminating seq. 4 4 - terminating seq. 5
Start printing with page #	2	152	0 to 9999
End printing after page #	2	154	0 to 9999
Start page number with	2	156	0 to 9999
Start headings with page #	2	158	0 to 9999
Start footings with page #	2	160	0 to 9999

INTEGRATE DATA AND TEXT INTO A DOCUMENT - EXPANDED TABLE

This function replaces variables in a document with data from an application or text from another document or glossary. The document is specified by the input file name and may be a standalone document or a document owned by MiniWord. All data variables and their values must be defined using ToolKit functions 50 and/or 51.

MAIN USES:

To integrate, from an application, the name and address from an inquiry screen with a form letter created in MiniWord. The procedure would be as follows:

1. Create the form letter in MiniWord. Mark the name and address information as variable. This document functions as the master copy and is not changed by the calling application.

2. The application conducts the inquiry. If the user wishes to generate a letter from the information on the screen, the application would request that the ToolKit create a working copy of the form letter from the master copy of the form letter.

3. The application would then repeatedly call ToolKit function 50 to define the variables and their values for the ToolKit.

4. The next step requires the application to integrate the defined values into the working copy of the form letter by using ToolKit function 40.

5. At this point the application could process the working copy of the letter as needed, i.e. print it, store it, and so forth.

See *Table 24*.

Table 24

Description	Length	Offset	Values
Function	1	0	40 - integrate data and text into a document
Reserved	7	1	
Input file type	1	8	32 - document 34 - glossary
Input file interface	1	8	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Output file type	1	132	ignored
Output file interface	1	133	ignored
Output file name	120	136	ignored

CONCATENATE TWO DOCUMENTS -

EXPANDED TABLE

This function will append a document or glossary to another document or glossary. The documents are specified by document name 1 and document name 2 and may be standalone documents or documents owned by MiniWord. Document 2 is appended to document 1. The main use of this function is to assemble a document from many other smaller documents.

See *Table 25*.

Description	Length	Offset	Values
Function	1	0	41 - concatenate two documents
Reserved	7	1	
Document 1 file type	1	8	32 - document 34 - glossary
Document 1 file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For document 1 file interface = 32:			
Document 1 file name	120	12	
For document 1 file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Document 2 file type	1	132	32 - document 34 - glossary
Document 2 file interface	1	133	32 - standalone file 33 - document owned by MiniWord
For document 2 file interface = 32:			
Document 2 file name	120	136	
For document 2 file interface = 33:			
Document name supplied?	1	136	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	138	
Folder name	60	162	
Password	8	222	
For document name supplied = 33:			
Optional menu title	64	138	

DELETE A DOCUMENT - EXPANDED TABLE

This function will delete the document specified by the input file name. You can delete either a standalone document or a document owned by MiniWord. See *Table 26*.

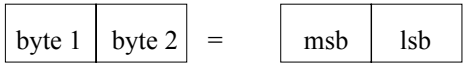
Table 26

Description	Length	Offset	Values
Function	1	0	42 - delete
Reserved	7	1	
Input file type	1	8	32 - document 34 - glossary
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	
Output file type	1	132	ignored
Output file interface	1	133	ignored
Output file name	120	136	ignored

SET NEW FILE OR DOCUMENT PARAMETERS - EXPANDED TABLE

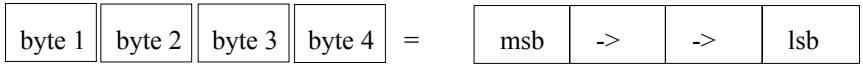
This function sets the parameters for new documents and ASCII files. You can set all the parameters individually or you can use the default parameters already set in MiniWord. The ToolKit uses the parameters set by this function whenever it creates new documents or ASCII files.

The format for all parameters which represent numbers and are two byte long (two-byte numerics) must have the most significant bits of the number in byte one and the least significant bits of the number in byte two, as illustrated:



An example of a two-byte numeric is the “Start printing with page #” parameter.

The format for all parameters which represent numbers and are four bytes long (four byte numerics) must have the most significant bits of the number in byte one, the next most significant bits in bytes two and three, and the least significant bits of the number in byte four, as diagramed below:



An example of a four-byte numeric is the “File size” parameter.

See *Table 27*.

Table 27

Description	Length	Offset	Values
Function	1	0	48 - set new ASCII file or document parameters
Reserved	7	1	32 - document
File type	1	8	33 - ASCII
For file type document:			
Use original defaults?	1	9	0- yes 1 - no
Read access	1	12	0 - creator 1 - all users
Write access	1	13	0 - creator 1 - all users
Delete access	1	14	0 - creator 1 - all users
Printer name	8	15	
Print mode	1	23	0 - continuous 1 - single sheet 2 - pause 1st page 3 - no form feeds 4 - twin sheet feeder
Copies	1	24	1 to 99
Left margin	1	25	0 to 99
Line spacing	1	26	0 - single spacing 1 - double spacing
Font	1	27	0 - draft 1 - letter quality 2 - proportional

Description	Length	Offset	Values
Global initial sequence #	1	28	0 - initial sequence 1 1 - initial sequence 2 2 - initial sequence 3 3 - initial sequence 4 4 - initial sequence 5
Global terminating seq. #	1	29	0 - terminating seq. 1 1 - terminating seq. 2 2 - terminiating seq. 3 3 - terminiating seq. 4 4 - terminating seq. 5
Start printing with page #	2	30	0 to 9999
End printing after page #	2	32	0 to 9999
Start page number with	2	34	0 to 9999
Start headings on page #	2	36	0 to 9999
Start footings on page #	2	38	0 to 9999
Lines per page	1	40	1 to 99
Top margin	1	41	0 to 99
Bottom margin	1	42	0 to 99
Pitch	1	43	0 - 10 cpi 1 - 12 cpi 2 - 15 cpi
Lines per inch	1	44	0 - 4 lpi 1 - 4.4 lpi 2 - 4.8 lpi 3 - 5.3 lpi 4 - 6 lpi 5 - 6.8 lpi 6 - 8 lpi 7 - 9.6 lpi 8 - 12 lpi
Page initial sequence #	1	45	0 - initial sequence 1 1 - initial sequence 2 2 - initial sequence 3 3 - initial sequence 4 4 - inital sequence 5

Description	Length	Offset	Values
Page terminating seq. #	146	46	0 - terminating seq. 1 1 - terminating seq. 2 2 - terminating seq. 3 3 - terminating seq. 4 4 - terminating seq. 5
Ruler left margin column	1	47	1 to 198
Ruler left margin type	1	48	0 - single space 1 - double space
Ruler right margin column	1	49	2 to 199
Ruler right margin type	1	50	0 - ragged 1 - justified
Tab 1 column	1	51	between L and R margins
Tab 1 type	1	52	0 - normal 1 - wrap 2 - decimal
Tab 2 column	1	53	between L and R margins
Tab 2 type	1	54	0 - normal 1 - wrap 2 - decimal
Tab 3 column	1	55	between L and R margins
Tab 3 type	1	56	0 - normal 1 - wrap 2 - decimal
Tab 4 column	1	57	between L and R margins
Tab 4 type	1	58	0 - normal 1 - wrap 2 - decimal
Tab 5 column	1	59	between L and R margins
Tab 5 type	1	60	0 - normal 1 - wrap 2 - decimal
Tab 6 column	1	61	between L and R margins
Tab 6 type	1	62	0 - normal 1 - wrap 2 - decimal

Description	Length	Offset	Values
Tab 7 column	1	63	between L and R margins
Tab 7 type	1	64	0 - normal 1 - wrap 2 - decimal
Tab 8 column	1	65	between L and R margins
Tab 8 type	1	66	0 - normal 1 - wrap 2 - decimal
Subject	60	70	
Written for	30	130	
For file type ASCII:			
Use original defaults	1	9	0 - yes 1 - no
Record length	2	12	1 to 255
File size	4	14	1 to operating system limit

SET LIST FILE OR LIST DOCUMENT -

EXPANDED TABLE

This function sets the name of the list file or list document. The list may be contained in an ASCII file, a standalone document, or a document owned by MiniWord. This function must be called before the ToolKit uses the interactive or background list processing functions.

The format of an ASCII list file is similar to the format of a MiniWord list document. Both the list file and the list document are divided into related groups of data items called records. From a user's standpoint the only significant difference between the list document and list file is the separator between records. Inside a MiniWord list document you must use a PAGE mark, CTRL-P. Inside an ASCII list file you must use the asterisk character, *, on a line by itself.

The first record in a list file or document is called the template. This record consists entirely of variable names. Each variable must be on a line by itself. The order of the variables in the template is important. The List Processing function correlates the data items from the other records in the file with the variable names in the template based on relative position within a record. For example, if the second line of the template contained the variable LASTNAME, then the second line of all the other records in the ASCII list file should have a last name in that line. If, for some reason, you are missing information when you create your list document, be sure to insert a blank line for the missing data item. Otherwise, the substitution of a data item for a variable during list processing will not be correct.

See *Table 29*.

Table 29

Description	Length	Offset	Values
Function	1	0	49 - set list file or document list parameters
Reserved	7	1	
Input file type	1	8	32 - document 33 - ASCII
For input file type = 33:			
Input file name	120	12	ASCII file name
Input file interface	1	9	32 - standalone file 33 - document owned by MiniWord
For input file interface = 32:			
Input file name	120	12	
For input file interface = 33:			
Document name supplied?	1	12	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	14	
Folder name	60	38	
Password	8	98	
For document name supplied = 33:			
Optional menu title	64	14	

SET DATA VARIABLE - EXPANDED TABLE

This function defines a variable and its value. It is used in conjunction with ToolKit function 40 (Integrate Data and Text into a document.) You may define up to 32 variables. To change the value of a variable , set it again with the new value. To erase all variables, set the variable name to all blanks. See *Table 30*.

Table 30

Description	Length	Offset	Values
Function	1	0	50 - set data variable
Reserved	7	1	
Data variable name	16	8	
Data variable value	128	24	

SET TEXT VARIABLE - EXPANDED TABLE

This function defines a variable and the document or glossary that contains the value of the variable. It is used in conjunction with ToolKit function 40 (Integrate Data and Text into a document). You may define up to 32 variables. To change the value of a variable, set it again with the new document or glossary name. To erase all variables, set the variable name to all blanks. See *Table 31*.

Table 31

Description	Length	Offset	Values
Function	1	0	51 - set text variable
Reserved	7	1	
Text variable name	16	8	
File type	1	24	32 - document 34 - glossary
File interface	1	25	32 - standalone file 33 - document owned by MiniWord
For file interface = 32:			
File name	120	26	
For file interface = 33:			
Document name supplied?	1	26	32 - yes 33 - no
For document name supplied = 32:			
Document name	24	28	
Folder name	60	52	
Password	8	112	
For document name supplied = 33:			
Optional menu title	64	28	

SET EDITOR PARAMETERS - EXPANDED TABLE

This function sets the parameters used by the ToolKit editor. This gives the application calling the ToolKit the ability to selectively disable editing functions, invoke the editor with up to 16 initial commands, and set the editor maximum right margin. See *Table 32*.

Table 32

Description	Length	Offset	Values
Function	1	0	52 - set editor parms
Reserved	7	1	
Editor functions enable bit map	8	8	see Note 1
Initial editor commands	16	16	see Note 2
Maximum right margin	1	33	2 to 199

Note 1: The editor functions enable bit map as follows:

Byte	Bit	Function	Byte	Bit	Function
0	0	Insert	2	0	Heading
0	1	Delete	2	1	Footing
0	2	Move	2	2	Column Counter
0	3	Copy	2	3	Command
0	4	Center	2	4	Help
0	5	Format	2	5	Bold
0	6	Scratchpad	2	6	Underline
0	7	Paragraph	2	7	Subscript
1	0	Page	3	0	Superscript
1	1	Spell Check	3	1	Overstrike
1	2	Document Insert	3	2	Variable
1	3	Search and Replace	3	3	User Enhancement
1	4	Hyphen	3	4	Tab
1	5	Alternate Character	3	5	Go To Page
1	6	Document Parameters	3	6	Typing Mode
1	7	Page Parameters	3	7	Reserved, set to 1

To enable a function, set its corresponding bit to 1.
Bytes 4, 5, 6, and 7 are reserved and should be set to all 1's.

Note 2: Up to 16 initial commands can be given to the editor to perform when it is started. If all 16 bytes are not needed, set any unused bytes to 0. The values for the commands are as follows:

Value	Command	Value	Command
1	Insert	25	Superscript
2	Delete	26	Overstrike
3	Move	27	Variable
4	Copy	28	User Enhancement
5	Center	29	Advance
6	Format	30	Backup
7	Scratchpad	31	Word
8	Paragraph	32	Line
9	Page	33	Sentence
10	Spell Check	34	Tab
11	Document Insert	35	Screen
12	Search and Replace	36	Up
13	Hyphen	37	Down
14	Alternate Character	38	Backspace
15	Document Parameters	39	Right
16	Page Parameters	40	Home
17	Heading	41	Bottom Home
18	Footing	42	Start Line
19	Column Counter	43	End Line
20	Command	44	Next Screen
21	Help	45	Previous Screen
22	Bold	46	Go to Page
23	Underline	47	Typing Mode
24	Subscript	48	End

EXIT - EXPANDED TABLE

This function terminates the ToolKit. Before continuing with its own processing, the application program must close the two communication files and wait for the ToolKit and all the ToolKit's child processes to terminate. See *Table 35*.

Table 35

Description	Length	Offset	Values
Function	1	0	127 - exit

Minisoft

Minisoft, Inc.
1024 First street
Snohomish, WA 98290
U.S.A.

1-800-682-0200
360-568-6602
Fax: 360-568-2923

Minisoft Marketing AG
Papiermühleweg 1
Postfach 107
Ch-6048 Horw
Switzerland

Phone: +41-41-340 23 20
Fax: +41-41-340 38 66
www.Minisoft.ch

Internet access:
sales@Minisoft.com
support@Minisoft.com
http//www.Minisoft.com
ftp://ftp.Minisoft.com