

---

# **Programmer's Guide**

# Contents

Programmers Guide .....	1
Contents .....	2
Source Libraries .....	7
File Locations .....	7
Global (common) Directories .....	8
Platform Specific Directories .....	8
Library Directories .....	8
Libraries .....	8
Naming Conventions .....	8
Guidelines for new libraries .....	10
Building Libraries .....	11
Overview .....	11
Software Used for PC Platforms .....	11
Compiler, Linker, Librarian, Resource Compiler .....	11
Make Utility .....	12
Heap Management .....	12
Font Scaling and Conversions .....	13
DBase (DBF) File Manager .....	14
Spelling-Checker Engine .....	14
Data Grid Display and Editor .....	15
SQL Dynamic Library .....	15
Docucorp Utilities .....	16
Software Used for MVS Libraries .....	17
Compiler and Linker .....	17
Runtime Library .....	17
Software Used for UNIX Libraries .....	18
Compiler and Linker .....	18
Make .....	18
SQL Dynamic Library .....	18
Third Party Source Modification Policy .....	18
Windows 9x/NT Requirements .....	19
Building a Library for PC Platforms .....	20
Switches and Settings .....	20
Configuring Make .....	21
Syntax of MAKEFILE.PRG and MASTER.PRG .....	23
Required Settings in MAKEFILE.PRG .....	24

Other Settings .....	24
Sample MAKEFILE.PRG.....	25
Running a Library Build .....	26
Adding Custom Code to a Library Make .....	27
Adding Libraries to Link .....	27
Custom Flags and Variables .....	27
Adding source files to a library.....	28
Batch Build for PC Platforms .....	28
The MK??? batch files .....	28
Files Used by MK??? .....	29
Files Created by MK???.....	30
Generating new MK??? and CMP??? files .....	30
System Basics .....	31
Commonly used System Data Types .....	31
VMMHANDLE .....	31
FAPPARM and FSIPARM .....	31
FAPPFN.....	32
FAP Object Message Handlers .....	32
FAPHANDLER Prototype.....	32
Virtual Memory .....	33
Linked Lists .....	33
Dynamic Arrays .....	38
Hashed Tables.....	40
Cache Management.....	42
Customizing the System .....	43
Print.....	43
Print Callback Functions.....	43
Support for Docusave .....	43
Support for OnDemand.....	45
Library Version Control.....	45
INI Settings .....	46
Functions.....	46
Customizing Batch Processing .....	47
CUSLIB .....	47
Base Rules .....	47
Image Rules .....	49
Field Rules .....	49
Recipient Rules .....	50
Customizing Documaker Workstation.....	52
Remote Access Library (RACLib and RacCo).....	52
Writing Custom Code .....	52
CSTLIB.....	52
Defining Custom Functions .....	52
Defining Custom Functions for Cross-Platforms.....	53
MENU Procedures.....	54
Menu Resource Format.....	54
Menu Keywords.....	54
Menu Item IDs .....	58
Menu Procedure Prototype .....	59
Menu Replacement .....	59
AFE Procedure Hooks .....	59
INI Options .....	60
Hook Prototypes .....	60
INI Settings .....	61
Functions and Hooks.....	61
Transactions .....	63

INI Definition .....	63
DAL Functions and Procedures .....	64
INI Registration .....	64
DAL Function Prototype.....	64
Edit Functions .....	65
Prototypes .....	65
Pre-Edit Functions .....	66
Post-Edit Functions.....	66
Image Functions.....	66
Prototypes .....	66
Open Functions .....	67
Close Functions.....	67
Export Formats .....	67
Import Formats .....	68
Document Set Procedures .....	68
INI Settings .....	69
Functions.....	69
Timed Service Functions .....	69
History .....	70
INI Settings.....	70
Example Registrations .....	73
Multiple Platforms .....	74
Timed Service Function Prototype .....	75
Messages .....	76
Considerations .....	77
Timing Example .....	77
Function and Hook Reference .....	79
AddComment.....	80
AFE Append Record Hook.....	81
AFERetriB4AppendgToLstHook .....	81
AFE Archive List Hook.....	82
AFERetDisplLstHook.....	82
AFE Archive Record Selected Hook .....	84
AFERetriOkButtonHook .....	84
AFE Check Form Set Data Hook.....	85
CheckUserEntry.....	85
AFE Complete Form Set Hook.....	86
Complete .....	86
AFE Entry Form Set Hook .....	88
EntryFormset .....	88
AFE Form Selection Buttons Hook .....	90
BUTTONx .....	90
AFE Initialization Hook.....	92
Init.....	92
AFE Parse Command Line Hook.....	93
Parse.....	93
AFE Post Edit Hook .....	94
PostEdit.....	94
AFE Pre Edit Hook .....	95
PreEdit .....	95
AFE Termination Hook .....	96
Term.....	96
AFE Window Procedure Hook .....	97
WindowProc .....	97
AFEArchive2WipKeys.....	98
Archive2WIP .....	98

AFESecurityFunc.....	99
Security .....	99
AFEWip2Archive .....	101
Wip2Archive.....	101
AFEWip2ArchiveRecord.....	102
Archive .....	102
AppIdxRec .....	103
CUSGetArcIdxName .....	104
IndexName.....	104
DSDefAppendBuffer .....	105
Append.....	105
DSDefCloseBuffer .....	106
Close .....	106
DSDefCreateBuffer .....	107
Create .....	107
DSDefFirstBuffer.....	108
First.....	108
DSDefNextBuffer .....	109
Next.....	109
DSDefOpenBuffer .....	110
Open.....	110
GVM.....	111
HaveGVM.....	112
LBYCARRetrieveFile .....	113
RetrieveFile.....	113
LBYCARRetrieveMemFile .....	114
RetrieveMemFile .....	114
LBYCARSaveFile .....	115
SaveFile .....	115
LMGLBYCheckin .....	116
CheckIn.....	116
LMGLBYCheckout .....	117
CheckOut .....	117
LMGLBYInit.....	118
Init.....	118
LMGLBYReInit.....	119
ReInit .....	119
LMGLBYSelect.....	120
Select.....	120
LMGLBYTerm.....	121
Term.....	121
LMGLBYUnlock.....	122
Unlock.....	122
LMGLBYView .....	123
View.....	123
MajorVersion.....	124
MinorVersion.....	125
PrinterClass .....	126
PrinterGroup .....	127
PRINT_IT .....	128
SetGVM.....	129
TMRTimers .....	130
TMRInit .....	131
TMRTerm .....	132
TMRSetAppData .....	133
TMRAppData .....	134

TMRSetHwnd .....	135
TMRHwnd .....	136
TMRSetHab .....	137
TMRHab .....	138
TMRIsDesktopUp .....	139
TMRIsDialogUp .....	140
TMRTimerTest1 .....	141
TMRTimerTest2 .....	142
TMRTimerTest3 .....	143
TRNAutoKeyIDUsrFunc .....	144
AutoKeyID .....	144
TRNSetBannerFormInfo .....	146
Set Banner Information .....	146

# Source Libraries

## File Locations

Each release has a directory tree that contains all of the files to produce that release. The directory tree is placed off the root of a drive and has a name that describes the release version. The following shows an example of version 11.1 with some of its directories.

\REL111	<i>Release number</i>
\DOC	<i>Documentation</i>
\3RDPARTY	<i>Third party file directories</i>
\RPS100	<i>Base library</i>
\INC	<i>Global include files used by all build platforms.</i>
\W32INC	<i>Platform global include files from third parties</i>
\W32LIB	<i>Platform import library files</i>
\W32BIN	<i>Platform debug binary files.</i>
\SHIPW32	<i>Platform ship binary files.</i>
\LIBNAME	<i>Source library directory (for example: FSILIB, FAPLIB, and IMGED)</i>
\C	<i>C and C++ source files</i>
\H	<i>Include files made for this library</i>

## Global (common) Directories

The Docucorp source is written to be compiled for many different target operating systems. For the most part, there is one set of source for the platforms. Libraries have their own source tree under the base library subdirectory (RPS100). Unique source and include files for a library are made and maintained in each library's subdirectory. When a library wants to expose some of its functionality to others, the relevant include files are copied to a global include directory (\INC). The files in the \INC directory are merely a copy of the source files that are maintained in the library subdirectory. These files are copied atomically with the build process.

## Platform Specific Directories

### *Include Directory*

Each PC operating system has its own include directory. This directory is used for platform specific include files. These files are generally from third parties. For example, the Windows 32-bit platform has some SQL include files in its W32INC directory.

### *Import Library Directory*

Each PC operating system has its own import library directory. This directory is where libraries get the files to link from other libraries. Both DLL import libraries and static libraries such as FSILIB are copied to this subdirectory for the target platform.

### *Binary Directory*

Each PC operating system has its own binary directory. This directory is where the debug version of the libraries is copied for the target platform.

### *Ship Directory*

Each PC operating system has its own ship files directory. This directory is where the non-debug version of the libraries is copied for the target platform.

## Library Directories

The subdirectory name should reflect the library name. Each library will have a subdirectory tree for source files, include files, and targets. The source file subdirectory is named "C", the header file subdirectory is name "H", and the targets are named by a three-letter acronym for the environment and a three letter acronym for the target file type.

## Libraries

### Naming Conventions

File names are generally created from abbreviations using the following syntax:

```
[Library][Editor Qualifier][Environment][Library Qualifier][.Extension]
```

Library file names are built from the library abbreviation and an abbreviation for the target environment, and in some cases, an editor or library qualifier is necessary.

Here are examples of LIB, DLL, and EXE names:



Library Target Name	Description
FSIW32.LIB	Core library for Windows 32-bit
VMMW32.DLL	Dynamic link library of VMM for Windows 32-bit
VMMW32.LIB	Import library of VMM for Windows 32-bit
INIW32.DLL	INI dynamic link library for Windows 32-bit
IMGEDW32.EXE	Image Editor for Windows 32-bit

### *[Library] Component*

Libraries should have a name that describes the function of the library and a two- or three-character abbreviation for that name. Here are examples of some currently established library names:

Library	Description
FAP	Forms Automation Platform
AFP	Advanced Function Print
VMM	Virtual Memory Management
GUI	Graphical User Interface
WIP	Work In Progress

### *[Editor Qualifier] Component*

Editor file names use an 'ED' as an editor qualifier. If a library is not an editor and not a conversion utility, then the editor qualifier is not used. Conversion utility names include a '2' before the target. Here are some examples:

Editor	Description
IMGED	Image Editor
LGOED	Logo Editor
PCL2FAP	Convert a PCL file into a FAP file.
FAP2DDT	Make a DDT file from a FAP file.

### *[Environment] names*

Environment names should have a name that describes the target platform in a two- or three-character abbreviation. Here are examples of environment names:

Environment	Description
W32	Windows NT/95/98/2000

### *[Library Qualifier] Component*

Here are examples of qualifiers:

Qualifier	Description
M	Migration
HX	Heap expander
C7	Microsoft C 7.0

### *[.Extensions] Component*

Here are examples of extensions:

Extension	Description
.LIB	A library. Can be a static linked library or an import library for a DLL.
.DLL	Dynamic link library
.EXE	Executable

## Guidelines for new libraries

### *Library Name*

The subdirectory name should reflect the library name. Each library will have a subdirectory tree for source files, include files, and targets. The source file subdirectory is named “C”, the header file subdirectory is name “H”, and the targets are named by a three letter acronym for the environment and a three letter acronym for the target file type.

### *Header Files*

#### Primary

Try to have only one header file for exported features of a library. This header should be copied to the \RPS100\INC directory by the make process.

#### Internal

If the library requires internal headers, make the internal headers include the primary header, not the other way around. Do not copy the internal headers to the INC directory.

#### Names

Try to name the header the same as the library. Here are examples:

```
VMM.H
INI.H
```

If there is some conflict with this name, then add LIB, for example,

```
FAPLIB.H
```

### *Source Files*

Try to name source files using the library abbreviation and something that describes the primary use of the source module. Here are examples:

```
FAPCURNT.C    Current access functions
INIUNLD.C     Unload INI file
FAPLDASC      Load ASC file
```

# Building Libraries

## Overview

Docucorp source is compiled for many operating systems. Only one set of source code is maintained for these many different target platforms. Source is maintained and built on the PC and uploaded to other non-PC platforms after successful completion of PC builds. The code has conditional compilation where necessary to handle the different operating systems. The core libraries isolate platform dependencies so libraries built upon these core ones rarely have to deal directly with conditional compilation in their own code.

## Software Used for PC Platforms

### Compiler, Linker, Librarian, Resource Compiler

The Windows platforms use the Microsoft Visual C++ compiler. The build process uses the compiler, linker, librarian, and resource compiler. However, the make program (NMAKE from Microsoft or MAKE from IBM) is not used by the Docucorp build process in favor of a more platform independent make program from Opus Software.

Software Version	Operating System Platform	Vendor Information
Microsoft Visual C++ version 7.0	Windows 32-bit	Microsoft Corporation <a href="http://www.microsoft.com">www.microsoft.com</a>

### *When this software is required*

If the libraries need to be re-compiled, you will need to get the compiler.

### *Using other software*

It is possible to use other software to compile and link system code. However, Docucorp does not recommend doing so. Generally...

- If you have a version of the Microsoft software that is later than the version used by Docucorp, then Docucorp libraries should be able to be linked without the need to recompile them.

- If you have another vendor's software, it is quite probable that you will need to re-compile the Docucorp libraries with your software unless the compiler supports setting that will generate code compatible with our libraries.

## Make Utility

Opus Make is the make system that Docucorp uses to build PC platforms. Opus Make allows Docucorp to maintain one make file for all the PC platforms the system is built on.

Software Version	Operating System Platform	Vendor Information
Opus Make version 6.12	Windows 32-bit	Opus Software, Inc. 1032 Irving St. Suite 439 San Francisco, CA 94122 (415) 485-9703 (800) 248-6787 <a href="http://www.opussoftware.com">www.opussoftware.com</a>

### *When this software is required*

Docucorp PC batch build process is set up to expect Opus Make.

### **MAKEFILE.PRG**

Each library has an Opus Make project file. This file defines the library type, libraries to link, and includes files to copy to a global directory. The MAKEFILE.PRG uses a MASTER.PRG file that handles the actual make process. The library MAKEFILE.PRG defines things for the MASTER.PRG file. Since the build settings and process flow are contained to one file, it ensures that all the libraries are built the same way.

## Heap Management

### *SmartHeap*

SmartHeap is the memory manager used by executable programs and is linked in to provide increased performance.

### *HeapAgent*

HeapAgent provides additional debugging facilities for Smart Heap. If you want to run the debug version of Docucorp software, you must acquire a license and include the heap agent file (HA312W32.DLL).

Software Version	Docucorp Platform	Vendor Information
SmartHeap Version 8.0	Windows 32-bit - (SHDW32M.LIB)	MicroQuill Software 11200 Kirkland Way Suite 310 Kirkland, WA 98033 (425) 827-7200

		<a href="http://www.microquill.com">www.microquill.com</a>
HeapAgent Version 3.1	Windows 32-bit - (HAW32M.LIB)	MicroQuill Software 11200 Kirkland Way Suite 310 Kirkland, WA 98033 (425) 827-7200 <a href="http://www.microquill.com">www.microquill.com</a>

### *When this software is required*

SmartHeap is not required to compile any Docucorp code; however, our build files are set up to use SmartHeap.

1. To use SmartHeap, the SmartHeap library must be the last library in the link list before the system libraries.
2. To use the standard C memory manager in place of SmartHeap, references to SmartHeap can be removed from the build process by modifying Docucorp MASTER.PRG. The MASTER.PRG has comments in it instructing where to remove references to SmartHeap for each operating system. To comment out a line in the MASTER.PRG, start the line with the # character.

For example, the following lines are in a WINDOWS 32-BIT section that activates SmartHeap and HeapAgent:

```
# ----- SmartHeap V8.0 library names -----
# Comment these names out if you do not want SmartHeap
    %if %null(ACTIVEX)
        %if %null(SMARTHEAP)
            SMARTHEAP=SHDW32M.LIB
        %endif
        %if %null(SMARTHEAPD)
            SMARTHEAPD=HAW32M.LIB
        %endif
    %endif
```

To eliminate SmartHeap and HeapAgent, comment out the lines like this:

```
# ----- SmartHeap V8.0 library names -----
# Comment these names out if you do not want SmartHeap
    %if %null(ACTIVEX)
        %if %null(SMARTHEAP)
            SMARTHEAP=SHDW32M.LIB
        %endif
        %if %null(SMARTHEAPD)
            SMARTHEAPD=HAW32M.LIB
        %endif
    %endif
```

## Font Scaling and Conversions

UFST is the dynamic linked library used for font conversions in Docucreate. UFST is contained within Docucorp's FXRLIB, and other Docucorp modules access UFST functions via FXRLIB.

Software Version	Docucorp Platform	Vendor Information
UFST Version 4.6 (Universal Font Scaling Technology)	Windows 32-bit - (IFW32.DLL, IFW32.LIB)	AGFA Corp. 100 Challenger Road Ridgefield Park, NJ 07660 (201) 440-2500

### *When this software is required*

Because UFST is isolated within FXRLIB, you do not need UFST unless FXRLIB is modified or re-compiled. Docucorp has modified the original UFST source code, so you must have a source code license from AGFA before you ask Docucorp to modify the source code on your behalf (see the [Third Party Source Modification Policy](#) section below).

## DBase (DBF) File Manager

CodeBase is the static library file manager used to provide access to dBase (DBF) files. CodeBase is contained within Docucorp's DB5LIB. Other Docucorp modules access CodeBase functions via DBLIB, which in turn calls DB5LIB.

Software Version	Docucorp Platform	Vendor Information
CodeBase Version 5.1	Windows 32-bit - (DB5W32.DLL)	Sequiter Software, Inc. PO Box 783 Greenland, NH 03840 (780) 437 - 2410 <a href="http://www.sequiter.com">www.sequiter.com</a>

### *When this software is required*

Because CodeBase is isolated within DB5LIB, you do not need to have CodeBase unless DB5LIB is modified or re-compiled. If DB5LIB needs to be re-compiled, you will need to get CodeBase from Sequiter Software. Docucorp has modified the original CodeBase source code. Since CodeBase has been modified by Docucorp, a source code license for CodeBase from Sequiter Software is required before requesting Docucorp to modify the source code on your behalf (see the [Third Party Source Modification Policy](#) section below).

## Spelling-Checker Engine

The Sentry Spelling-Checker Engine is a static library used by Docucorp software. The Sentry Spelling-Checker Engine is contained within Docucorp's SPLLIB, and other Docucorp modules access Spell Checker functions via SPLLIB.

Software Version	Docucorp Platform	Vendor Information
Sentry Spelling-Checker Engine Version 5.15	Windows 32-bit - (SPLW32.DLL)	Wintertree Software Inc. 69 Beddington Ave. Nepean, Ontario Canada K2J3N4 (613) 825 - 6271 (800) 340 - 8803 <a href="http://www.wintertreesoftware.com">www.wintertreesoftware.com</a>

### *When this software is required*

Because Sentry Spell-Checker Engine is isolated within SPLLIB, you do not need to have Sentry Spell-Checker unless SPLLIB is modified or re-compiled. Docucorp has modified the original Sentry Spelling-Checker Engine. Since the Sentry Spelling-Checker Engine has been modified by Docucorp, you must get a source code license from Wintertree Software before you ask Docucorp to modify the source code (see the [Third Party Source Modification Policy](#) section below).

## Data Grid Display and Editor

DataTable is a dynamic linked library that provides a “spreadsheet like” grid component that is used by Docucreate and the Entry module of Documaker Workstation. DataTable is contained within Docucorp’s DTBLLIB, and other Docucorp modules access the data table function via DTBLLIB.

Software Version	Docucorp Platform	Vendor Information
DataTable Version 3.0	Windows 32-bit - (DTBW32.DLL)	Proto View 2540 Route 130 Cranbury, NJ 08512 (609) 655-5000 <a href="http://www.protoview.com">www.protoview.com</a>

### *When this software is required*

Because DataTable is isolated within DTBLLIB, you do not need DataTable unless DTBLLIB is modified or re-compiled. Docucorp has modified the original DataTable source code. Since DataTable has been modified by Docucorp, you must get a source code license from ProtoView before you ask Docucorp to modify the source code on your behalf (see the [Third Party Source Modification Policy](#) section below).

## SQL Dynamic Library

The dynamic SQL library allows access to file systems via SQL statements. This SQL interface is contained within Docucorp’s DB2LIB or SQLLIB--depending upon the platform running. Other Docucorp modules access dynamic SQL functions via DBLIB, which in turn calls DB2LIB or SQLLIB.

Software Version	Docucorp Platform	Vendor Information
DB2 software developer kit Version 6.1	Windows 32-bit - (DB2W32.DLL) AIX - (DB2 6.1)	IBM Corp. <a href="http://www.ibm.com">www.ibm.com</a>
SQL Server Version 6.5	Windows 32-bit - (SQLW32.DLL) ODBC for NT 3.70.06.23 ODBC for 95 3.60.03.02	Microsoft Corporation <a href="http://www.microsoft.com">www.microsoft.com</a>
Oracle Server 7.3.2	HP-UX	Oracle Corporation <a href="http://www.oracle.com">www.oracle.com</a>

### *When this software is required*

Since SQL Dynamic Library is isolated within the libraries listed above, you do not need SQL Dynamic Library unless you are modifying or re-compiling one of the listed libraries.

## Docucorp Utilities

These utilities are used in the Docucorp build process. They are developed by Docucorp and are supplied with the source release.

### *SLEEP*

Makes the current process sleep for the number of seconds requested.

### *SYNC*

This utility synchronizes files and directories.

```
SKEMP Version 1.4 Jun 24 1997 11:06:32
SYNC path1 path2 [path3]
Sync files from path1 to path2
if path3 is specified the files are copied there.
/? This help info
/V Verify before copying files
/S Subdirectories included in file sync
/C Consolidate matching files into a single destination directory
/R Overwrite read-only files on destination
/H Include Hidden and System files in file sync
/N No Copy performed. Matching files simply identified
/2 2-Way files sync
/A Attributes preserved from source file
/D Delete destination file first to conserve space
/X No-Existing files on destination are copied from source
/U Update only existing files on destination
/F Force read-only on files copied to destination
/M Move files (source files deleted after copy)
/O Sync OLDER files over newer
/P Path derived from file, where /P=DestDriveLetter
/W Wait time if copy to destination fails and try again, where /W=seconds
/B Error Bells sound if sync fails, where /B=count
```

### *CCHK*

This is a C source checker for things that compilers for other environments do not support. It checks that lines are no longer than 72 characters. Forces the /\* \*/ style C comments...

```
CCHK Version 1.01 Jun 8 1999 15:03:34 by SKEMP
CCHK filePattern [filePattern...]
Search for illegal cross-platform items in text files
defaults search includes // ^ [ ] and TAB characters.
A ^ will indicate that a default search item was found.
A ! will indicate a line length exceeds maximum.
A $ will indicate an EOF marker was found.
A X will indicate a #include case error was found.
A * will indicate a user-entered item was found.
A M will indicate characters follow continuation char.
/? This help info
/S Subdirectories included in search
/I Input user items to search
/C Case insensitive search used
/L Line maximum= change default from 71
/V Verbose output
/N No Caret check
/F No Forward // characters checked
/M No line cont. \ characters checked
/T No Tab characters checked
/[ No square bracket characters checked
```



```
/X Exit program on first error
/Z End of File Check for Ctrl+Z
/# No #include case checked
/Q Quiet will suppress output messages
```

## **DDEL**

This utility deletes files and directories.

```
SKEMP Version 2.01 Jan  4 1996 10:19:00
DDEL pattern [pattern...]
Delete files in directory and subdirectories
/? This help info
/V Verify before deleting file
/S Subdirectories included in file search
/D Delete Subdirectories if empty, after deleting files
/R read-only files also deleted
/Z Zero Length files only deleted
```

## **DDIR**

This utility shows a directory of files and directories.

```
SKEMP Version 2.0 Apr 10 1995 11:17:53
DDIR pattern
Show files w/ attributes in directory and subdirectories
/? This help info
/S Subdirectories included in search
/A All files with attributes
/P Search Path for specified files
/E Error Level set if matching found
/Q Quiet output messages
```

# Software Used for MVS Libraries

See the **MVS Installation Guide** for more information on MVS requirements and configurations.

## Compiler and Linker

If you plan to customize your system by writing custom rules or other custom code, or if your license includes source code, you will also need one of the following compilers from IBM:

- C/370 version 2.1 or higher
- C/C++ For OS/390, version 2.10 or higher

Compile times are much faster with this compiler than with the other compilers. Additionally, we have observed approximately an 8% performance improvement in GenData and GenPrint runs when the C/C++ compiler has been used versus the other compilers. We recommend you use C/C++ For OS/390, version 2.10 (or higher) compiler.

## Runtime Library

One of the following runtime libraries from IBM Corp is required:

- C/370 Library Version 2.2 or higher
- Language Environment (LE) 1.5 or higher

Docucorp recommends using the language environment (LE) runtime.

## Software Used for UNIX Libraries

### Compiler and Linker

Software Version	Operating System Platform	Vendor Information
IBM C SET++ for AIX/6000, version 2, release 1	AIX	IBM Corp. <a href="http://www.ibm.com/">http://www.ibm.com/</a>
Sun Workshop C/C++ 5.0	Sun Solaris	Sun Microsystems <a href="http://www.sun.com/">http://www.sun.com/</a>
ANSI C Compiler Software 10.10	HP-UX	Hewlett-Packard <a href="http://www.hp.com/">http://www.hp.com/</a>

### Make

Standard UNIX make utility

### SQL Dynamic Library

The dynamic SQL library allows access to file systems via SQL statements. This SQL interface is contained within Docucorp's DB2LIB or SQLLIB--depending upon the platform running. Other Docucorp modules access dynamic SQL functions via DBLIB, which in turn calls DB2LIB or SQLLIB.

Software Version	Operating System Platform	Vendor Information
DB2 software developer kit Version 6.1	AIX	IBM Corp. <a href="http://www.ibm.com">www.ibm.com</a>
Oracle Server 7.3.2	HP-UX	Oracle Corporation <a href="http://www.oracle.com">www.oracle.com</a>

### *When this software is required*

Since SQL Dynamic Library is isolated within the libraries listed above, you do not need SQL Dynamic Library unless you are modifying or re-compiling one of the listed libraries.

## Third Party Source Modification Policy

Docucorp will not ship modified third party source code that it does not have the rights to release. Docucorp will make identical modifications to the original source code (as would a contractor) on behalf of the properly licensed customer or business partner, when requested to do so in writing. This makes sure Docucorp is complying with third party software license agreements, both in spirit and to the letter.

The procedures are:

1. Back up your original purchased software.
2. Send us the original media (CD, diskettes, and so on) for operating systems version, along with a letter requesting that we make any and all modifications necessary to be compatible with version 11.1.

When Docucorp receives the software and the request Docucorp will:

1. Copy the source code you provided to a hard drive and compare the directories and compare source file names and content, for both "C" and "H" files, to the version we maintain and establish a list of differing files.
2. Place the following comment block at the top of each source module requiring modification:
 

```

/*****
Source modifications performed by Docucorp International
at the request of Company Name.

MM/DD/YY Initials or programmer
*****/

```
3. Comment out all of the source code in the version you have provided in the following fashion:
 

```

#if 0
... original source
#endif

```
4. Place a comment before our source:
 

```

/*****
* Docucorp modifications begin:
*****/

```
5. Insert the source code that is compatible with the shipping version of version 11.1.  
 Note: Since there may be new modules with different software versions, those modules may be completely commented out without any code inserted to take its place.
6. Ensure that we are able to build the library and that it will work correctly with version 11.1.
7. Return the original software along with our modifications.

This complete process may take a little time, but will provide the results you are looking for. Following this procedure will be viewed as compliant with source code licensing restrictions of the software vendor, and preserve the necessary rights.

## Windows 9x/NT Requirements

Note: These instructions only apply to versions 10.3 and earlier.

Follow these steps to install Visual Studio .NET 2003.

1. Apply all service packs.
2. Set this environment variable:
 

```
VCTOOLS=\Program Files\Microsoft Visual Studio .NET 2003\Vc7
```
3. Go to this directory:
 

```
\Program Files\Microsoft Visual Studio .NET 2003\Vc7\src
```
4. Copy the \_sample\_.DEF and \_sample\_.RC files into the R10MSVC.DEF and R10MSVC.RC files.
5. Modify makefile by replacing all references to \_sample\_ with references to R10MSVC.
6. Modify R10MSVC.RC to indicate the company name and DLL name.
7. Modify R10MSVC.DEF to indicate the DLL name.
8. Run bldnt.cmd

The new R10MSVC.DLL and R10MSVC.LIB files are built in the \Program Files\Microsoft Visual Studio .NET 2003\Vc7\crt\src\build\intel directory.

## Building a Library for PC Platforms

### Switches and Settings

The switches and settings can be found in the MASTER.PRГ file in the base directory \REL111\RPS100.

#### *Windows 32-bit*

Compile Switch	Description
/nologo	Suppress copyright message
/W3	Set warning level (default n=1)
/Z7	Enable old-style debug info
/Od	Disable optimizations (default)
/G3	Optimize for 80386
/Zp1	Pack structures on n-byte boundary
/J	Default char type is unsigned
/MD	Link with MSVCRT.LIB
/D "WIN32"	Define
/D "_WINDOWS"	Define
/D _WIN32_IE=0x0200	Define
/D "WIN32_LEAN_AND_MEAN"	Define
/D "_DEBUG"	Define

Link Switch	Description
/NOLOGO	Suppress startup banner
/MACHINE:I386	Specifies the target platform
/NODEFAULTLIB:LIBCMT	Ignore specified library when resolving externals
/NODEFAULTLIB:MSVCRT	Ignore specified library when resolving externals
/NODEFAULTLIB:MSVCRTD	Ignore specified library when resolving externals
/SUBSYSTEM:windows,4.0	* Tells the operating system how to run the .EXE file. The WINDOWS subsystem applies to an application that does not require a console, probably because it creates its own windows for interaction with the user. If the variable QUICKWIN is empty, then this setting is used.
/SUBSYSTEM:console	* Tells the operating system how to run the .EXE file. The CONSOLE subsystem is for a Windows 32-bit character-mode application. If the variable QUICKWIN is not empty, this setting is used.

/DEF:\$(TARGET).def	Definition file
/DLL	* If DLL, then say so
/IMPLIB:\$(TARGET).lib	* Create an import library for DLL

### *Using the Microsoft IDE (and NMAKE)*

If you do not have Opus Make and wish to set up a library to use the Microsoft environment, you will have to set the switches that apply to the library on settings windows. All of the switches above that do not have an asterisk (\*) in the description apply to all libraries. (To find out special settings and link files for a library you must examine the MAKEFILE.PRG file for that library. This document explains settings and syntax of the MAKEFILE.PRG and the MASTER.PRG file that it includes. Refer to other areas of this document for more information on those topics.

/W3	Use warning level 3
/Gf	Enable string pooling
/Z7	Use C 7 style CodeView Info
/Od	Disable optimizations
/D "_DEBUG"	Define
/D "__WIN__"	Define
/GEf	Customize Windows entry/exit
/Gee	Customize Windows entry/exit
/D "__PPS__"	Define when making PPS
/GD	Protected mode Win entry/exit code
/AL	Large memory model
/GD	Protected mode Win entry/exit code
/Alw	Custom memory model
/Ged	Customize Windows entry/exit
/GA	Protected mode Win entry/exit code
/AL	Large memory model

## Configuring Make

Each Docucorp library has an Opus Make project ( MAKEFILE.PRG) file for each target platform supported by that library. This MAKEFILE.PRG is read by a "make makefile" utility (MKMF by Opus software) that generates a make file (MAKEFILE.) that is used by the make utility. This MAKEFILE has all the source files found in the library's C directory, their dependencies, compile instructions, link instructions, and file distribution instructions in it. The batch build process will automatically call the make makefile utility to generate a new make file when one does not exist.

The MAKEFILE.PRG is the road map for the library. It is set up so you can easily define the components of a library without having to get involved in all the details. The MAKEFILE.PRG includes a MASTER.PRG file that defines the specific instructions used to do the actual build. The master project uses the values defined in the individual libraries' MAKEFILE.PRG to build the library for the platform requested.

The Docucorp make process lets you override your C environment, compile directory, and certain file locations, without having to change the MASTER.PRG. Environment variables are used to define many of the components used in the make process. To override defaults, just define the environment variable before running Opus Make.

## *C Environment*

The environment variable CENVIRON sets the C environment. The variable is defaulted per platform so it is not usually necessary to set this variable.

Here is an example:

```
SET CENVIRON=MSDEV
```

The supported values for CENVIRON are MSDEV for Windows 32-bit.

## *Compiler Directory*

The environment variable CMPDIR sets the compiler base directory. The variable is defaulted per platform and CENVIRON so it may not be necessary to set this variable. To override the defaults, define this environment variable before running Opus Make. To make your changes more permanent, add them to your AUTOEXEC.BAT file.

Docucorp Platform	Default Value
Windows 32-bit	If msdevdir is not defined, then CMPDIR=\PROGRA~1\DEVSTU~1  The msdevdir variable is usually set up by the Microsoft compiler installation.

## *Other variables*

Other variables can be set before running Opus Make. These variables can be set in the AUTOEXEC.BAT file if they apply across libraries. If you want settings for only a particular library, add the setting to the library MAKEFILE.PRГ before the include of the MASTER.PRГ file.

Variable	Description
CFLAGS	Compiler flags
EXTRINC	Extra #includes only the first one will get /I
EXTRALIBS	Extra libraries to link
HCOPY	Header files to be copied
IDLFLAGS	IDL compiler flags
INCLUDE	Include file directories
LDFLAGS	Linker flags
MS32LIBRARIES	Microsoft Windows 32-bit libraries
RFLAGS	Resource compiler flags
XTRAPATH	Another path for the target

## *Docucorp Directories*

The Docucorp directory variables are defaulted so it may not be necessary to set these variables. To override the defaults, define the environment variable before running Opus Make. To make your changes more permanent, add them to your AUTOEXEC.BAT file.

Variable	Default
FSISYS	..\.
FSIINC	+= \$(FSISYS)\INC
FSILIB	\$(FSISYS)\W32LIB (for Windows 32-bit)
FSIBIN	\$(FSISYS)\W32BIN (for Windows 32-bit)
FSISHP	\$(FSISYS)\SHIPW32 (for Windows 32-bit)

## Syntax of MAKEFILE.PRG and MASTER.PRG

A brief description of some of the syntax will be covered here. If you do not have Opus Make, this information should help you read these files to make settings in your environment. If you do have Opus, you can find out more about this in your Opus documentation.

### Overview

The Opus syntax is similar to the C language syntax. For example, one equal sign does an assignment, two equal signs do a comparison, and exclamation equal sign is not equal. Commands are preceded by a percent sign (%) such as %if which is similar to C's pre-processor commands such as #if.

### Comments

Comments begin with a pound sign on a line. The comments only apply to text following the pound sign. They do not automatically span lines. Therefore, a pound sign should be placed before each line of text that is a comment.

Here are some examples:

```
##### ASSIGN THE APPROPRIATE ANSWERS IN THIS SECTION UNTIL
##### YOU REACH STOP
RESULT      = DLL           # Choose DLL EXE or LIB
```

### Branching Commands (%if, %else, %elif, and %endif)

Opus has syntax for if blocks. The %if requires a %endif to complete the block. A %else and %elif can also be used within the block.

Here is an example of an if block using all the if block commands:

```
%if $(OS) == OS2           # If compiling for OS/2
    TARGET = FAPOS2
%elif $(OS) == MSDOS       # Else if compiling for Win16
    TARGET = FAPWIN
%else                      # Otherwise forcing Windows 32-bit
    TARGET = FAPW32
%endif
```

### Other Commands (%null, %exists)

Opus provides a %null command to test if a variable is set.

Here is an example:

```
%if %null(TARGET)          # If the target variable is not set
    %abort Nothing to build
%endif
```

There is also a %exists command to test if something exists.

```
%if %exists(..\..\..\doc)  # If there is a doc directory
```

```
copy $(TARGET).DOC ..\..\..\doc
%endif
```

## *Variables*

Variables can be defined at any time. Use the equal sign to assign a value to a variable. The following is an example of setting up a variable called result:

```
RESULT      = DLL                # Choose DLL EXE or LIB
```

To access the value of a variable use a dollar sign followed by parentheses surrounding the variable name.

Here is an example:

```
%if $(RESULT) == LIB
    # do something
%endif
```

## *Variable Lists*

To make a variable list use the plus then the equals sign.

Here is an example:

```
FSILIBS  = FSIOS2.LIB
FSILIBS += VMMOS2.LIB
```

## Required Settings in MAKEFILE.PRG

### *RESULT*

The RESULT variable must be set in the libraries MAKEFILE.PRG. This variable tells the build process what type of target to make. RESULT can be set to DLL, LIB, or EXE.

Here is an example:

```
RESULT      = DLL                # Choose DLL EXE or LIB
```

### *TARGET*

The TARGET variable must be set in the libraries MAKEFILE.PRG. This variable tells the build process what to name target file. The file name uses both the TARGET and RESULT to make the name.

Here is an example:

```
TARGET      = CUSW32            # base name
```

### *MASTERPRG*

The MASTERPRG variable must be set in the libraries MAKEFILE.PRG. This variable tells the build process what file to include as the master project file. Almost all the libraries have the same setting for MASTERPRG in their MAKEFILE.PRG file.

Here is an example:

```
MASTERPRG=..\..\MASTER.PRG
```

## Other Settings

All other settings are optional or a default value is supplied. The following is a list of settings that are usually set in the library MAKEFILE.PRG.



Variable	Description
MULTICOMP	Multiple platforms compile. This means the library is compiled for a non-PC platform such as MVS. If this variable is set, run the CCHK utility to make sure the file has no invalid syntax and can be uploaded to the host machine.
QUICKWIN	If this variable is set, then the Windows EXE is to be QUICKWIN. This is usually set for non-GUI utility programs.
STOPERR	If this variable is set, the build will stop when an ERR file exists.
VERSRC	If there is a library version file for this library, then the file name (without extension) should be set for this variable. The library version module will always compile.
FSILIBS	List of Docucorp libraries to link. Enter the file name and extension.
HCOPY	List of header files to copy to the global INC directory. Supply the file name and extension.

## Sample MAKEFILE.PRG

Here's the MAKEFILE.PRG for CUSLIB:

```
# -----
# MKMF program settings

#!!!!!!! ASSIGN THE APPROPRIATE ANSWERS IN THIS SECTION UNTIL
#!!!!!!! YOU REACH STOP
RESULT      = DLL           # Choose DLL EXE or LIB
MULTICOMP   = Y             # multi platform compile
STOPERR     = Y             # Stop if an ERR file exist

CCHKCMD     = /n            # Do no do caret check

# -----
# Program names
#
%if $(OS) == WIN32
    TARGET   = CUSW32        # library name for Windows 32-bit
%endif

# -----
# Library Version Module to always compile
# - No extension please.
#
VERSRC = CUSVERSN

# -----
# FSI Libraries used for this program
# - Do not enter a path, (FSILIB) will be added automatically
# - Remember we have at least two platforms for many projects

%if $(OS) == WIN32
    FSILIBS  = FSIW32.LIB
    FSILIBS += VMMW32.LIB
    FSILIBS += INIW32.LIB
    FSILIBS += FAPW32.LIB
    FSILIBS += RPW32.LIB
    FSILIBS += GLBW32.LIB
    FSILIBS += RCBW32.LIB
    FSILIBS += GVMW32.LIB
    FSILIBS += DSW32.LIB
    FSILIBS += DBHW32.LIB
    FSILIBS += DALW32.LIB
```

```

        FSILIBS += GENW32.LIB
        FSILIBS += ASCW32.LIB
        FSILIBS += RULW32.LIB
        FSILIBS += UTLW32.LIB
        FSILIBS += PRTW32.LIB
        FSILIBS += LOGW32.LIB
        FSILIBS += RULW32.LIB
        FSILIBS += W32FIX.LIB
%endif

# -----
# Header files to copy
#   Assign each file that should be copied to the INC directory
#   - path (LOCINC) will be used so don't assign a path.

HCOPY  = CUSLIB.H

#####
#!!!!!!! STOP
#!!!!!!! IN MOST CASES YOU WILL NOT HAVE TO MAKE ANY CHANGES BELOW THIS LINE
#!!!!!!!
# -----
# MKMF automatically maintains the HDRS, EXTHDRS, OBJS and SRCS macros.
# The EXTHDRS, HDRS, and SRCS macro are not explicitly used in this makefile
# but you may have a need for them elsewhere, such as for revision control.
#
# get source and object by like extensions i.e. *.obj *.res

%if %null(SRCS)
SRCS.RC      =
OBJ.S.RES    =
OBJ.S.OBJ    =
EXTHDRS      =
HDRS         =
SRCS         =
OBJ.S        =
%endif

MASTERPRG=..\..\MASTER.PRG
%include $(MASTERPRG)

```

## Running a Library Build

To build a library, you must change directory to the libraries target directory, such as W32DLL, W32LIB, or W32EXE. In that directory the MAKEFILE.PRG and MAKEFILE. file for the library will be found.

---

If you do not have a MAKEFILE. file, you must run MKMF to generate one.

---

To run MKMF type:

Command	Platform
MKMF OS=WIN32	Windows 32-bit

Once a MAKEFILE. has been generated for a library, MAKE can be run.

To run MAKE type:

Command	Platform
MAKE OS=WIN32	Windows 32-bit

For example, to build the Windows 32-bit version of CUSLIB you would:

1. CD \REL111\RPS100\CUSLIB\W32DLL
2. MKMF OS=WIN32 *(if there is no MAKFILE. Possibly using OS2MKMF on NT.)*
3. MAKE OS=WIN32 *(Possibly using OS2MAKE on NT.)*

## Adding Custom Code to a Library Make

The Microsoft C run time DLL built with MS Visual Studio 6.0 is required to be able to compile the Documaker Server version 11.1.

### Adding Libraries to Link

#### *Docucorp Library*

Docucorp libraries are defined in the FSILIBS variable in the MAKEFILE.PRG for a library. To make modifications, you must edit the MAKEFILE.PRG for that library changing the FSILIBS variable. Once the change has been made, you must run the MKMF utility before running MAKE.

#### *External Library*

To add external libraries to a library link set the EXTRALIBS variable in the MAKEFILE.PRG. To make modifications, you must edit the MAKEFILE.PRG for that library changing the EXTRALIBS variable. Once the change has been made, you must run the MKMF utility before running MAKE.

The following example adds SQL\_DYN to a link:

```
EXTRALIBS=\SQLLIB\SQL_DYN.LIB
```

### Custom Flags and Variables

You can customize a library build by setting any of the following variables in the MAKEFILE.PRG. Once the change has been made, you must run the MKMF utility before running MAKE.

Variable	Description
CFLAGS	Compiler flags
EXTRINC	Extra #includes only the first one will get /I
EXTRALIBS	Extra libraries to link
HCOPY	Header files to be copied
IDLFLAGS	IDL compiler flags
LDFLAGS	Linker flags
MS32LIBRARIES	Microsoft Windows 32-bit libraries
RFLAGS	Resource compiler flags
XTRAPATH	Another path for the target

## Adding source files to a library

You can add your own source files to a library. The library directory structure will look like this:

```
\LIBNAME  Source library directory (for example: FSILIB, FAPLIB, and IMGED)
  \C      C and C++ source files
  \H      Include files made for this library
  \W32LIB Static library target directory.
  \W32DLL Dynamic library target.
  \W32EXE Executable library target.
```

To add your new source:

1. Copy your new header files to the H directory
2. Copy your new source files to the C directory
3. Go to the target directory (W32DLL, W32EXE, ...)
4. Delete the MAKEFILE.
5. Run MKMF to create a new make file (for Windows 32-bit, type MKMF OS=WIN32).
6. Run MAKE to create a new library (for Windows 32-bit, type MAKE OS=WIN32).

## Batch Build for PC Platforms

The batch build process is a series of batch command files and utilities that runs through the release directory building the requested libraries. This process makes many assumptions about your environment. It uses the third party software described above and, most importantly, requires Opus Make.

### The MK??? batch files

The MK??? batch files have been set up to run builds for Windows 32-bit compiles.

The ??? is replaced by the target (W32). Therefore, the following batch files are provided for builds.

MKW32.BAT builds the Windows 32-bit platform

Run these command files from the \REL111\RPS100 directory.

### Syntax

MKW32 [qa]make/[qa]link/[qa]all/[qa]force [RESTARTDIR] [flags]

Param1: Use **'make'** to make only those modules that have changed

Use **'link'** to delete all .EXE and .DLL files, and remake

Use **'all'** to rebuild makefile, delete all objects, and remake

Use **'force'** to compile all targets

Add **'qa'** to the beginning of make/link/all to also copy the release to the QARELEASE drive R:

Param2: Use **'RESTARTDIR'** to restart the make at a given directory, such as FAPLib, and continue the batch build from there.

Param3: Optional Opus Make settings

### *Examples:*

To run a normal Windows 32-bit build you would type:

```
MKW32 make
```

To run a Windows 32-bit build that copies to QARELEASE drive you would type:

```
MKW32 qamake
```

## Files Used by MK???

### *INPROC.BAT*

This batch file is called to generate an INPROC.??? file when a build is run. If MKW32 is run, then INPROC.W32 is generated. When MK??? is run it checks to see if its INPROC.??? exists. If INPROC.??? exists then it stops and reports who is running the build by typing INPROC.???.

---

If you have had an abnormal end of your build (loss of network, Ctrl+Break...) then be sure to delete the INPROC.??? file you generated in the \REL111\RPS100 directory. Also check for \*.TMP files in the \REL111\RPS100 directory. Type them and if you created them, then they must be deleted. As the build is running, a .TMP is generated for the library being compiled. When the build tries to compile a library and there is a .TMP by that library name, the build process sleeps for a while then tries to create the .TMP file again. By doing this the MK??? makes sure there is never more than one operating system accessing the files of a library. It synchronizes the compiles somewhat across operating system builds.

---

### *ALLFILES.BAT (or ALLFILES.CMD)*

This file provides the list of libraries to build. The ALLFILES.BAT file is the master file so for OS2 and VIS builds this file is copied to ALLFILES.CMD. To add a library to the build you must check out ALLFILES.BAT and make an entry in it for the new library name. It is important to make sure you add your new library in the proper order. The DLL and LIB files are built in the order of dependency for other libraries. The EXE files are listed alphabetically followed by PMSC specific stuff and FSIVER.EXE.

### *CMP???.BAT (or CMP???.CMD)*

This command file compiles the library. The ALLFILES files calls this command file passing in the library name and compile switches.

### *ERR.BAT*

This command file echoes \*.ERR files into \REL111\RPS100\ERRORS.???.

### *UPDDOC.BAT*

This command file updates both the RES&DEV and the QARELEASE drive with the latest document files.

### *UPDHLP.BAT*

This command file updates both the RES&DEV and the QARELEASE drive with the latest Help files.

## *CLEANQA.BAT*

Deletes the files and subdirectories on the QARELEASE drive for RPS100, PMSC, INSTALL, MVS, and DOC.

## Files Created by MK???

### *INPROC.???*

The \REL111\RPS100\INPROC.??? file is generated when the MK???. See INPROC.BAT above for more information.

### *???????.TMP*

The \REL111\RPS100\??????.TMP file is generated as the library name .TMP for the library being built by the MK??? command file.

## Generating new MK??? and CMP??? files

### *GENMKS.BAT*

The GENMKS.BAT file generates the MK??? and CMP??? files. If you need to make a change to the MK??? or the CMP??? file then you should do it in this file so that command files for each operating system can be easily made and re-created.

---

If you do not make your changes in this file also, then your changes can be overwritten at a later point by someone running the genmks batch file.

---

Syntax: GENMKS OS2/WIN/W32/VIS  
Param1: Specify Operating system W32  
When W32 it generates MKW32.BAT and CMPW32.BAT  
Example:  
GENMKS W32

# System Basics

This section covers general customizations that can be done for batch rules processing or for Documaker Workstation. Refer to the sections [Customizing Batch Processing](#) or [Customizing Documaker Workstation](#) for specifics.

## Commonly used System Data Types

### VMMHANDLE

A *VMMHANDLE* is a virtual memory handle that in many environments merely resolves to a standard pointer reference. The definition for *VMMHANDLE* is found in *FSI.H*.

### FAPPARM and FSIPARM

The variable type *FAPPARM* is defined in *UTL.H* as a redefinition of *FSIPARM*.

```
#define FAPPARM FSIPARM
```

The *FSIPARM* definition is found in *FSI.H*. A *FSIPARM* needs to be a variable that can handle several variable types that may need to be passed to the handler. For most environments, a *DWORD* is sufficient storage space and for others (OS/400) a definition is required and takes the following form:

```
typedef FSIPARMTYPE _FSIPARM
{
    VMMHANDLE vmmh;
    VOID FAR *ptr;
    FAPPFN     fn;
    DWORD      dw;
    WORD       w;
    BYTE       b;
} FSIPARM;
```

Macros have been provided to access the different variable types passed as a *FAPPARM/FSIPARM*. You should use these macros to access variable so that your custom code will always work if changes are made to the *FSIPARM* definitions for your platform.

```
FAPPARM2VMMH(p1)      // Access a VMMHANDLE
FAPPARM2PTR(p1)        // Access a void pointer (void *)
FAPPARM2PFN(p1)        // Access a function pointer
FAPPARM2DWORD(p1)      // Access a DWORD
FAPPARM2WORD(p1)       // Access a WORD
```

```

FAPPARM2BYTE(p1)          // Access a BYTE
FAPPARMDEF                // Default parameter value (0)
FAPPARMVMMH(p1)           // Pass a VMMHANDLE
FAPPARMPTR(p1)            // Pass a void pointer (void *)
FAPPARMPFN(p1)            // Pass a function pointer
FAPPARMDW(p1)             // Pass a DWORD
FAPPARMW(p1)              // Pass a WORD
FAPPARMB(p1)              // Pass a BYTE

```

### Example

```

DWORD _VMMAPI MyFieldHandler(VMMHANDLE fieldH,
                             DWORD      msg,
                             FAPPARM    p1,
                             FAPPARM    p2)
{
    FAPWINDOW * fw;
    . . .
    switch (msg) {
    case FAP_MSGATTRIBUTES:
        fw = (FAPWINDOW*) FAPPARM2PTR(p1); // Access the pointer
        FAPSendObjectMessage(fieldH,
                              FAP_MSGSELECT,
                              FAPPARMPTR(fw), // Pass a pointer
                              FAPPARMDEF);    // Pass a default value

        break;
    . . .
    }
    return(OrigFieldHandler(fieldH,msg,p1,p2));
}

```

### FAPPFN

A *FAPPFN* is a pointer to a void function. The *FAPPFN* definition is found in *FSI.H*.

```
typedef void (_VMMAPIPTR FAPPFN)(void);
```

If the value represents some other function prototype, an appropriate cast will be required to call the procedure or assign it to another variable.

## FAP Object Message Handlers

Each *FAPOBJECT* (defined in *FAPFORM.H* and structures defined in *FAPDEF.H*) has a registered message handler that acts similar to the way window procedures handle messages for a window. The registered message handler for an object reacts to messages sent within the system to perform the requested task for the object. You can install your own object message handler to intercept messages and perform your customizations. The handlers conform to the *FAPHANDLER* prototype.

### FAPHANDLER Prototype

This prototype is defined in *FAPFORM.H* and takes the following form:

```

typedef DWORD (_VMMAPIPTR FAPHANDLER)(VMMHANDLE objectH,
                                       DWORD msgno,
                                       FAPPARM p1,
                                       FAPPARM p2);

```

Name	Description
ObjectH	Represents a VMMHANDLE to a FAPOBJECT.



Msgno	Contains the specific message number being passed to the function. Each type of message used by FAPOBJECTs must be unique. There is a list of pre-defined messages in FAPFORM.H, but this list may be extended by defining your own messages with FAP_MSGUSER + <i>n</i> , where <i>n</i> represents some number greater than zero.
p1	May contain values to be used by the functions and are specific to the message number being passed.
p2	May contain values to be used by the functions and are specific to the message number being passed.

### Example

```

FAPHANDLER OrigFieldHandler; // Need to keep original handler
. . .
OrigFieldHandler = FAPSetObjectHandler(FAP_OBJFIELD, MyFieldHandler);
// Set my field handler and save the original
. . .
// After I am through I will restore the original handler
FAPSetObjectHandler(FAP_OBJFIELD, OrigFieldHandler);
. . .
//-----
DWORD _VMMAPI MyFieldHandler(VMMHANDLE fieldH,
                             DWORD      msg,
                             FAPPARM    p1,
                             FAPPARM    p2)
{
    FAPWINDOW * fw;
    . . .
    switch (msg) {
    case FAP_MSGATTRIBUTES:
        fw = (FAPWINDOW*) FAPPARM2PTR(p1); // Access the pointer
        FAPSendObjectMessage(fieldH,
                              FAP_MSGSELECT,
                              FAPPARMPTR(fw), // Pass a pointer
                              FAPPARMDEF);    // Pass a default value

        break;
        . . .
    }
    return(OrigFieldHandler(fieldH,msg,p1,p2)); // Call the original
}

```

## Virtual Memory

The Virtual Memory Management library (VMM) offers a set of functions for managing a large amount of data using a limited amount of conventional memory. Data is managed in structures: doubly linked lists (ordered and unordered), hashed symbol tables, data caches, and dynamic arrays. The system utilizes the memory management functionality that VmmLib provides so you need to become familiar with this library in order to manipulate things within the system. For information on the Docucorp API, refer to the online API documentation found with the release [REL111\DOC\API\INDEX.HTM](#).

### Linked Lists

Doubly linked list routines create and manage chains of variable-length records in virtual memory. A list consists of a list descriptor and a chain of variable-length elements. Each element contains data and is preceded by an element header.

The doubly linked list data structure is a valuable, general-purpose method for managing a related set of data, particularly when:

- The number of data elements is not known in advance.
- The allocated space for the data elements does not need to be contiguous.
- The size and structure of a data element may vary.
- The makeup of the data set is volatile (lots of additions and deletions.)
- The data set has an order that must be maintained as elements are inserted or deleted.

## *Handles*

Each descriptor or element is identified by a handle. A handle, represented by the data type `VMMHANDLE`, contains a value that uniquely identifies a given list or element. The content of a handle is managed by library functions. Do not attempt to directly manipulate the contents of a handle.

Implementations prior to version 8.0 used a four-byte unsigned long integer as a handle (version 8.0 and up uses a far void pointer), but applications should not rely on this. Always use the `sizeof` directive to compute the size of a handle. Currently, if it is necessary to "printf" a handle, use `%lu`, `%ld`, `%lx`, or `%lp`.

## *List Descriptors*

A list descriptor "describes" a virtual doubly linked list. List descriptors are allocated in virtual memory and contain the following information:

- The handle of the list itself.
- The handle of the first element in the list. To obtain, use `VMMFirstElem`.
- The handle of the last element in the list. To obtain, use `VMMLastElem`.
- The number of elements currently stored in the list. To obtain, use `VMMCountList`.
- The total number of elements that has been added to (or inserted into) the list. To obtain, use `VMMTotAddList`.
- The offset of key data within an element data record. To obtain, use `VMMKeyOffset`.
- The number of key components. To obtain, use `VMMKeyCount`.
- The key comparison function. To obtain, use `VMMKeyCompare`.
- The type of the list:
  - `VMMLST_NORMAL = 0`,
  - `VMMLST_ORDERED = 1`,
  - `VMMLST_HASHSYM = 2`,
  - `VMMLST_FROZEN = 3`.
- To obtain, use `VMMLSTType`.
- The number of hash buckets. To obtain, use `VMMHashBuckets`.
- The handle of the hash bucket table (or the frozen-list handle table). To obtain, use `VMMHashTable`.
- List descriptors are identified by a handle. The content of a list descriptor is managed by library functions. Do not attempt to directly address or manipulate the content of a list descriptor.

## *Elements*

An element contains data that has been inserted into a list. An element can be up to 64K in length.

For DOS and Heap Expander: An element can be up to about 16K in length. Although as much as 32 megabytes of data can be stored, the number of elements that a list can contain is limited by available conventional memory. (There is a physical hardware limit of 64K handles, and the actual limit is much less.)

Elements are allocated in virtual memory and consist of the data and a set of control information (called an element header) which includes the following:

The handle of the list itself. In this way, each element "knows" the list to which it belongs. To obtain, use VMMElemList.

The handle of the next element in the list. To obtain, use VMMNextElem.

The handle of the previous element in the list. To obtain, use VMMPrevElem.

The length of the element (the number of bytes of data stored in the element.) To obtain, use VMMElemLength.

The hash bucket number of the element, or if it is a frozen list, the element index. To obtain, use VMMHashBucket.

Elements are identified by a handle. The content of an element header is managed by library functions. Do not attempt to directly access or manipulate the content of an element header.

## *Creating a List*

Before a list can be used, it must be created. The create process will assign a handle which is used to identify the list in subsequent call to library functions. To create a list, use VMMCreateList.

## *Destroying a List*

When a list is no longer needed, its allocated memory can be released by destroying the list. Once a list has been destroyed, its handle is no longer valid. To be used again, the list must be recreated. To destroy a list, use VMMDestroyList. Destroying a list also releases the memory used by any elements in the list, and any memory used to create a hashed symbol table for the list.

## *Inserting Elements*

Elements are added to a list via an insertion process. Every element points both forward and backward. In order to insert a new element, an existing element must be identified as an insertion point. An element designated as an insertion point will become the next element in the chain following the new element about to be inserted.

For example, suppose a list contains elements A, B, and D. Assume we wish to insert element C between B and D. To accomplish the insert correctly, we must name D as the insertion point element.

In some linked list schemes, the first element points backward to NULL and the last element points forward to NULL. In the VMM library, the list descriptor is actually part of the chain. When a list is empty, the first and last elements point to the list itself. When elements are added, the first element points backward to the list, and the last element points forward to the list. Therefore, to insert an element at the end of the list, the list descriptor itself is named as the insertion point element.

To add an element to the end of a list, use VMMAppendElem. To insert an element at the front of a list, use VMMPushElem. To insert an element into any position in a list, use VMMInsertElem. To insert an element into the proper position in an ordered list, use VMMInsOrdElem.

## *Deleting Elements*

Any element can be deleted from a list. When an element is deleted from somewhere in the middle, the forward and backward pointers of the elements before and after it are corrected to take up the slack. Once deleted, the space allocated to an element is returned to the resource pool. To delete an individual element, use `VMMDeleteElem`. To delete the first element in a list, use `VMMPopElem`. To delete all of the elements in a list, use `VMMFreeList`. To free all elements in a list and destroy the list itself, use `VMMDestroyList`.

## *Navigating a Linked List*

Linked lists can be sequentially processed (forward or backward) from any point in the list. All that is needed is the handle of the element at which to begin. Forward navigation is accomplished using `VMMNextElem`. Backward navigation is managed with `VMMPrevElem`. Both functions return a handle. This handle can be used to retrieve the element.

The end of the list has been reached when the handle of an element is the same as the handle of the list (because the list descriptor itself is part of the chain.)

There are numerous ways of obtaining a starting point. Most library functions return an element handle. Any handle can be used as a starting point. Typically, however, most sequential processing starts at the beginning and goes forward to the end. For

In addition, as shown earlier, a more efficient (but not so straightforward) method makes use of the fact that the last element in a list is the element prior to the list descriptor. Once understood, use this as the preferred form:

```
for ( elemH = listH;
      (elemH = VMMPrevElem(elemH) != listH; ) {
    // etc.
```

Sometimes it is desirable to view a list as an unbroken circle without the list descriptor to get in the way. The functions `VMMNextCircElem` and `VMMPrevCircElem` are similar to the above except that the handle of the list descriptor is never returned. In this case, the assumption is that some mechanism other than the end of the list will be used to determine when to stop processing.

## *Accessing Elements*

Elements are retrieved via `VMMGetElem`, and stored via `VMMPutElem`.

DOS note: A temporary pointer to the current location of the element's data can be retrieved via `VMMElemPtr`. It is important to realize that this is a temporary pointer. Temporary pointers are only valid until the next VMM library function call. The physical location of any element is dynamic. Each time an element is accessed, it is located via its handle and temporarily made addressable via a process called handle de-referencing.

As more library and application code is built using the VMM library, it becomes prudent to regard any function as a potential call to the VMM library. Therefore, the point cannot be stressed enough that care must be taken in the use of temporary pointers. For example:

```
typedef struct {
    char      field1 DIM(20);
    int field2;
    long      field3;
} MY_STRUCT;
```

```
MY_STRUCT    myStruct;
MY_STRUCT    *ptr;
VMMHANDLE    elemH;
VMMHANDLE    listH;
int          length;
```

```

ptr = VMMElemPtr(elemH);

// "ptr" is now valid until the next function
// that calls the VMM library (potentially any
// function.) Until then, we can use "ptr" as
// much as we like.

strcpy(ptr->field1,"hello world");
ptr->field2 = 123;
ptr->field3 = 456;

VMMAppendElem(listH,&myStruct,sizeof(MY_STRUCT));

// We must now assume that "ptr" is invalid. It might
// still be valid, but we have no easy way to know,
// and we can only safely assume that it no longer
// points to the data contained in element "elemH".
// In order to use "ptr" again, it must first be
// refreshed.

ptr = VMMElemPtr(elemH);

```

## *Ordered Lists*

An ordered linked list is one that is kept in an ascending order according to values in a key position in the record area of the elements.

The information necessary to establish a key for an ordered linked list can be set with `VMMKeyOrdList`. Once the key information has been defined, elements can be added in order using `VMMInsOrdElem`. Elements can be retrieved from an ordered list using `VMMLocateOrdElem`. Both functions make use of the lower level utility function `VMMFindOrdElem`.

In some cases, building an ordered list is one means of accomplishing a sort. To specifically sort a list, use `VMMSortOrdList`, which will use the key information established by `VMMKeyOrdList` and use a lower level function `VMMQSort`.

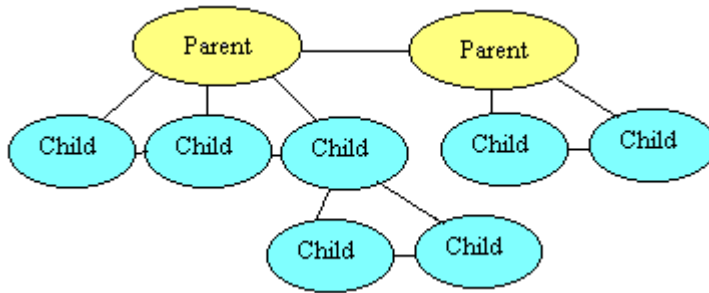
## *Frozen Lists*

Versions 9.0 and above provide support for "frozen" lists, to help improve overall system performance. A frozen list is an ordered list that has been marked as "frozen" and then optimized internally for faster access. The assumption is that once a list is stable, i.e., no longer needs to be modified, a snapshot of the state of the list can be taken and saved. This snapshot process is called "freezing" the list. When a list is frozen, a table of handles is created which provides easy indexing to any element in the list. Functions that locate elements in ordered lists by key or index operate at dramatically faster rates. The time required to freeze a list is generally negligible next to the time it would otherwise take to locate elements in a normal, non-frozen ordered list. If a frozen list is subsequently modified, `VMMLIB` will automatically "unfreeze" the list, and then it becomes simply an ordered list once more.

A list can be explicitly frozen and unfrozen with `VMMFreezeList` and `VMMUnfreezeList`. The function `VMMSortOrdList` implicitly freezes an ordered list automatically.

## *Node-linked lists*

A node-linked list is one that is lists within lists thus creating a collection. A tree is created by using the node functions.



Each of the nodes named parent above are elements in a list. They have children that are elements in their own list. Each child itself could be a parent with children.

## *Using a List as a Queue*

A linked list can be utilized as a queue. Queues are processed FIFO, or "first in first out." New elements are put into the queue by always inserting them at the end using VMMAAppendElem. The front of the queue is the element returned by VMMAFirstElem. When that element has been processed, it can be deleted with VMMADeleteElem, which will then return the next element.

## *Using a List as a Stack*

A linked list can also be utilized as a stack. Stacks are processed LIFO, or "last in first out." New elements are pushed onto the stack by inserting them at the beginning. This process is simplified by using VMMPushElem. Once processed, the top of stack is deleted (or popped off) using VMMPopElem. Any element can be floated to the top of the stack by using VMMAFloatElem. Any element can be sunk to the bottom of the stack by using VMMSinkElem. Any element can be moved to any other position in the list using VMMAMoveElem.

## **Dynamic Arrays**

The Dynamic Array functions build on top of the same lower level virtual memory functions used by the linked list functions. A dynamic array trades some of the flexibility of the linked list in exchange for less overhead and a greater number of possible elements.

Dynamic arrays should be used instead of linked lists when:

- The array metaphor seems a better fit.
- The elements are relatively short and fixed in length.
- Elements do not need to be inserted or deleted, but rather can be simply pigeonholed into a slot via an index.

## *The VMMAARRAY Structure*

Dynamic arrays are identified by a VMMAARRAY structure. The address of the structure is passed to the manipulation functions. Handles are only used internally. Any given element can be retrieved via an index. To support more than 64K elements, indexes are declared along (unsigned long). The contents of the VMMAARRAY structure may be examined, but only library functions should be allowed to modify the variables.

## *Initializing an Array*

Arrays are initialized via `VMMInitArray`. At initialization time, the element length and a blocking type number are used to compute a block size and a blocking factor. Elements are stored in blocks of virtual memory. Handle overhead is reduced by maximizing the number of elements stored per block.

Block types are numbers that represent the approximate size of the block. For example, a block type of 2 represents a block size of about 2K bytes. Valid block types are 1, 2, 4, 8, and 16. Restricting to these block sizes ensures the best use will be made of the 16K virtual page.

## *Conventional Memory Usage*

Dynamic arrays allocate an array block of handles in conventional memory. If conventional memory is not available, an attempt to add a dynamic array block will fail. If this happens, a function pointer is examined. By default, VMM will invoke its own dynamic array "out of conventional memory" function which will display the message "VMMDefNoFreeMem:nnnn" (where nnnn is the amount of memory requested), and exit the program. As the program exits, the virtual memory system will be terminated.

A different "no free memory" function may be installed, or the function can be completely disabled, at any time.

## *Array bounds checking*

At initialization time, a maximum number of elements can be specified. If the maximum elements parameter is set to zero (use the equate `VMMNO_BOUNDS`), an array's size is limited only by the amount of conventional memory available to keep track of the virtual handles and pages. If a non-zero value is specified, representing the maximum number of elements that can be accessed, array bounds checking is enabled.

If bounds checking is enabled, each time an array is accessed, the index will be validated. If the maximum is exceeded, a function pointer is examined. If the pointer is not zero (`VMMBADINDEX` or `NULL`), a "bad index function" is invoked. By default, VMM will invoke its own bad index function that will display the message "VMMDefBadIndex:nnnnnn" (where nnnnnn is the invalid index), and exit the program. As the program exits, the virtual memory system will be terminated.

A different bad index function may be installed, or the function can be completely disabled, at any time. For example:

```
VMMSetBadIndex( (VMMBADINDEX) NULL );
```

Will disable the bad index function. In this case, arrays with maximum elements specified will be bounds checked, but all invalid indexes will be mapped to the highest valid index. An application can install its own bad index function as follows:

```
void *_VMMAPI MyBadIndexFunc(VMMARRAY *array, ulong index,void *rec);
```

```
. . .
```

```
VMMSetBadIndex(MyBadIndexFunc);
```

## *Freeing an Array*

Individual elements of an array cannot be freed. All of an array's elements are freed at once using `VMMFreeArray`.

Once freed, an array can be used again without re-initializing. The previously set values for the element length, blocking factor, and upper bounds are preserved. If a new geometry for the array is desired, the array should be re-initialized with `VMMInitArray` using new values.

## *Accessing an Array Element*

Array elements are accessed via an index using `VMMGetArray`, `VMMPutArray`, and `VMMArrayPtr`. These functions return a temporary pointer to the element. A temporary pointer may be `NULL` if the out-of-memory function is disabled and memory was exhausted while attempting to allocate a block to contain the element. If not `NULL`, a temporary pointer is valid until the next call to any VMM library function. The same warnings about temporary pointers that were expressed in the section on "Accessing Elements" for linked lists, also applies to dynamic arrays.

## **Hashed Tables**

A symbol table is a collection of data elements that are accessed by name. Typically, the list is large, the order of the elements in the list is not important, and the time to retrieve an element by symbol name must be as short as possible.

VMM hashed symbol tables use an efficient hashing algorithm on a variable length null-terminated string to produce an integer value. This integer value (called a bucket number) is used as an index into an array of `VMMHANDLEs`. Each handle in the array (called a bucket) is the first element in a list of elements that share the same bucket number. This much smaller list can quickly be searched for the desired element. By reducing the number of compare operations, and eliminating most of the movement through the list, the time to access an element by key is dramatically reduced.

### *Initializing a hashed symbol table*

Any linked list can become a hashed symbol table if it has a null-terminated string field that can be used as a key. As usual, a list is created with `VMMCreateList`. It can be converted to a hashed symbol table at any time after that by calling `VMMInitHashList`. If the list already contains elements when `VMMInitHashList` is called, the elements are converted and any duplicates are eliminated.

### *Freeing a hashed symbol table*

When a hashed symbol table is initialized with `VMMInitHashList`, an array of `VMMHANDLEs` is created. This array is known as the hash bucket table. The size of the bucket table is determined by the `numBuckets` parameter passed to `VMMInitHashList`. In addition, at initialization type, the internal type of the list is set to `VMMLST_HASHSYM` (2). If the list is no longer needed, it can be destroyed with `VMMDestroyList`. If the list needs to stay around, but no longer needs to be used as a hashed symbol table, the virtual memory used by the bucket table can be released by calling `VMMFreeHashList`. This also resets the list type to either `VMMLST_NORMAL` (0), or `VMMLST_ORDERED` (1).

### *Inserting elements into a hashed symbol table*

Once a hashed symbol table has been initialized, elements can be inserted via `VMMInsertHashElem`. The internal hashing algorithm determines the bucket to which the new element belongs. If that bucket is empty, the element is appended to the list and its handle is assigned to that bucket. If the bucket already has one or more elements, the new element is inserted into the list next to those elements. Each element in the hashed symbol table has recorded in its element header the number of the bucket to which it is assigned. The list of elements that belong to that bucket is searched sequentially. When an element is found that does not belong to that bucket, the element is inserted ahead of it. If a duplicate symbol entry is found, the new element overlays the existing element.

### *Deleting elements from a hashed symbol table*

Elements are deleted from a hashed symbol table with `VMMDeleteHashElem`.



### *Locating elements in a hashed symbol table*

The handle of an element can be retrieved by key from a hashed symbol table with VMMLocateHashElem. This function uses a lower level function VMFindHashElem that searches a specified bucket.

### *Updating elements in a hashed symbol table*

Once the handle of an element is obtained, the data of the element can be accessed with VMGetElem, VMMElemPtr, and modified with VMMPutElem and VMMPutElem. If, however, any change is made to the data of an element that affects either the value of the key field, or the size of the element, the element must be updated with VMUpdateHashElem. Changing the size of the element will require a new handle for the element. Changing the key value will more than likely affect the bucket number of the element. In either case, if VMUpdateHashElem is not used, the integrity of the list could be compromised.

### *Comparing ordered lists, frozen lists, and hashed symbol tables*

The following charts help illustrate the performance comparisons between the three types of lists that support keyed access.

The statistics show the relative speed of the list search algorithms. All of the tests were run under the OS/2 environment. (Note that in order to obtain meaningful numbers, the number of calls had to be increased as the key size was reduced.)

Key length of 1000 bytes. 100,000 calls.

Elements in list	10	100	500	1000
Hash list	23	27	36	36
Frozen list	15	30	44	51
Ordered list	15	35	64	95

Key length of 100 bytes. 300,000 calls.

Elements in list	10	100	500	1000
Hash list	9	10	13	13
Frozen list	9	15	17	18
Ordered list	9	18	48	86

Key length of 50 bytes. 300,000 calls.

Elements in list	10	100	500	1000
Hash list	6	7	8	8
Frozen list	6	9	11	13
Ordered list	6	14	43	77

Key length of 10 bytes. 500,000 calls.

Elements in list	10	100	500	1000
Hash list	7	8	9	10
Frozen list	9	13	15	16
Ordered list	9	19	60	124

## Cache Management

A cache is a collection of data elements that are pooled for possible reuse. Cache support is implemented via hash tables and dynamic arrays.

# Customizing the System

## Print

### Print Callback Functions

#### *Prototypes*

The print callback function prototypes are found in the PRTLIB.H file. There is a function prototype macro *clbck\_func\_def*. The defines are as follows:

```
#define clbck_func_def(fname) \
        DWORD _VMMAPI fname(VMMHANDLE formsetH, \
                                DWORD msg, \
                                FAPPARM p1, \
                                FAPPARM p2)
```

### Support for Docusave

Docusave can archive AFP and Metacode print streams. To produce print streams that can be archived by Docusave, the print streams must be in a Docusave-compatible format and must contain special records used to index the archive. The OutMode option in the Metacode or AFP print control group will cause the Metacode or AFP print stream to be written in a Docusave-compatible format. There are two Docusave-compatible formats that are supported, “MRG2” and “MRG4”.

For example,

```
< PRTType:AFP >
OutMode = MRG4

< PRTType:MET >
OutMode = MRG2
```

When OutMode is set to “MRG4”, print stream records will have a 4-byte sequence that precedes them defining their length. Records will be grouped into blocks and there may be one or more records within a “block”. Both records and blocks have a 4-byte sequence that precedes them defining their length. These length indicators are formed by taking the high-order byte of length followed by the low-order byte of length followed by two bytes of zeros. Thus, the maximum number that can be displayed is a 16-bit quantity. The value in each includes the length of the structure itself. A one-byte data record in its own block would have 5 for the record length and 9 for the block length. The following shows what a 3-byte record would look like:

Byte Offset	Value (Hex)	Meaning
0	00	Block length High-Order
1	0B	Block length Low-Order
2	00	Always 0
3	00	Always 0
4	00	Record length High-Order
5	07	Record length Low-Order
6	00	Always 0
7	00	Always 0
8	31	'1'
9	32	'2'
10	33	'3'

“MRG2” uses a subset of the above blocking scheme. It consists a two-byte record length preceding each record. Again, the value contains the length of the header itself. It was designed for the PC type of system where the low-order byte of length is first followed by the high-order value. The example record above would look like this:

Byte Offset	Value (Hex)	Meaning
0	05	Record length Low Order
1	00	Record length High Order
2	31	'1'
3	32	'2'
4	33	'3'

In addition to using OutMode to produce the print streams in a Docusave-compatible format, special records must be included in the print streams in order to index the archive. These special records are written into the print stream as comment records. A DAL script can be used to add these comment records into the print stream. A DocuSaveScript option in the Metacode or AFP print control group will cause a DAL script to be executed at the times when Docusave comments can be added to the print streams. To add Docusave comments to an AFP print stream, you would need to add the DocuSaveScript option, containing the name of a DAL script to execute.

For example,

```
< PrtType:AFP >
DocuSaveScript = Docusave.DAL
OutMode = MRG4
```

Additional DAL functions have been added to assist in creating archive keys to use with Docusave.

Function	Description
<a href="#">AddComment</a>	Adds a comment string to the print stream
<a href="#">ApplIdxRec</a>	Gets an archive record based on APPIDX.DFD and Trigger2Archive INI settings
<a href="#">HaveGVM</a>	Verifies if a GVM variable exists
<a href="#">SetGVM</a>	Updates the contents of a GVM variable

<a href="#">GVM</a>	Gets the contents of a GVM variable
<a href="#">MajorVersion</a>	Gets the system's major version number
<a href="#">MinorVersion</a>	Gets the system's minor version number
<a href="#">PrinterClass</a>	Gets the type of print being produced
<a href="#">PrinterGroup</a>	Gets the name of the print group being used
<a href="#">Print_It</a>	Debug tool to print a string to the console

## Support for OnDemand

OnDemand is part of an IBM suite of products that provide high performance document capture, indexing, storage, retrieval and presentation of AFP data streams. To enable a Documaker Server-produced AFP print stream to be archived by OnDemand, you will need to define a new INI setting and provide a DAL script that produces the archive keys to use.

In the printer control group set up for AFP printing (usually `PrtType:AFP`), add an the `OnDemandScript` option with the name of a DAL script you want the system to execute.

For example,

```
< PrtType:AFP >
  OnDemandScript = OnDemand.DAL
```

Additional DAL functions have been added to CUSLIB to assist in creating archive keys to use with OnDemand.

Function	Description
<a href="#">AddComment</a>	Adds a comment string to the print stream
<a href="#">AppldxRec</a>	Gets an archive record based on APPIDX.DFD and Trigger2Archive INI settings
<a href="#">HaveGVM</a>	Verifies if a GVM variable exists
<a href="#">SetGVM</a>	Updates the contents of a GVM variable
<a href="#">GVM</a>	Gets the contents of a GVM variable
<a href="#">MajorVersion</a>	Gets the system's major version number
<a href="#">MinorVersion</a>	Gets the system's minor version number
<a href="#">PrinterClass</a>	Gets the type of print being produced
<a href="#">PrinterGroup</a>	Gets the name of the print group being used
<a href="#">Print_It</a>	Debug tool to print a string to the console

## Library Version Control

The version-control code is replaced through INI options that specify what function to call when certain operations are requested. All these options are under the group "VCS". "Wrapper function" shows the version-control function that should be called from another library. This function checks the given INI option to see what function to call, using the given default if no INI option is specified. The defaults do version-control operations through LBYLIB. (The same way it worked in previous versions.)

## INI Settings

```
[ VCS ]
Checkin = LMGOS2->LMGLBYCheckin
Checkout = LMGOS2->LMGLBYCheckout
Init = LMGOS2->LMGLBYInit
ReInit = LMGOS2->LMGLBYReInit
RetrieveFile = LBYOS2->LBYCARRetrieveFile
RetrieveMemFile = LBYOS2->LBYCARRetrieveMemFile
SaveFile = LBYOS2->LBYCARSaveFile
Select = LMGOS2->LMGLBYSelect
Term = LMGOS2->LMGLBYTerm
Unlock = LMGOS2->LMGLBYUnlock
View = LMGOS2->LMGLBYView
```

## Functions

You can replace two sets of functions. You can replace the LMG functions to completely replace the version-control code, from dialog boxes. If you do this, you need not worry about the LBY functions- they'll never be called. You can also leave the default LMG functions, but replace the LBY functions- This will use most of the default VCS code, only replacing the part where files are actually stored/retrieved from some kind of database. The standard windows will still be used, a database of version, user ID, comments, and so on will be kept in the usual way, but the actual data can be stored in a different repository.

To customize version-control, just provide replacements for the desired functions, and set the INI options so that the new functions are called. More details (parameters, and so on.) are documented in the online API documentation of LMGLIB and LBYLIB found with the release [REL111\DOC\API\INDEX.HTM](#), but here's a brief list of what the functions do:

Functionality	Description
Init	Any one-time initialization that needs to be done. To find out more, see the <a href="#">LMGLBYInit</a> section.
ReInit	Any re-initialization that needs to be done after the master-resources has been changed. (The default LMGLBY functions need a chance to re-read some INI options, and so on.) To find out more, see the <a href="#">LMGLBYReInit</a> section.
Term	One-time cleanup when program terminates. To find out more, see the <a href="#">LMGLBYTerm</a> section.
Select	Allows user to select a file without taking any other action. To find out more, see the <a href="#">LMGLBYTerm</a> section.
Checkout, View, Checkin, Unlock	Called when user selects the matching menu item. To find out more, see the <a href="#">LMGLBYCheckout</a> , <a href="#">LMGLBYView</a> , <a href="#">LMGLBYCheckin</a> , and <a href="#">LMGLBYUnlock</a> sections
SaveFile	Saves a file to some sort of repository, returning a key to it. To find out more, see the <a href="#">LBYCARSaveFile</a> section.
RetrieveFile	Retrieves a file from repository to disk, given its key. To find out more, see the <a href="#">LBYCARSaveFile</a> section.
RetrieveMemFile	Retrieves file directly into memory. To find out more, see the <a href="#">LBYCARRetrieveMemFile</a> section.

# Customizing Batch Processing

## CUSLIB

The CUSLIB library is where you should make customizations for a batch system. For more information on processing rules, refer to the [Rules Processor System Guide](#) and the [Rules Reference](#). For information on the Docucorp API, refer to the online API documentation found with the release [\REL111\DOC\API\INDEX.HTM](#).

### Base Rules

Base rules are stored in the Job Definition Table (AFGJOB.JDT). This file is used to define job level (level 1) and form set level (level 2) rules.

#### *Job level rules (level 1)*

Job level rules are global rules used to apply procedures prior to and following the processing of all transactions in a given set. Most of these rules are designed to initialize processing; open and close necessary data files; allocate or release resources used during processing; and other specialized functions do exist.

#### *Form set level rules (level 2)*

These rules are also known as transaction rules. Such rules are designed to manage and manipulate the behavior of form sets. Form set level rules typically affect the form set as a whole. These rules are responsible for the following:

- includes functions that initialize or reinitialize resources between transactions
- determines which transactions to include or exclude from the run
- controls the creation of the form set
- controls what happens to the form set after the transaction has completed

## Prototypes

The base rule prototypes are found in the RPLIB.H file. There is a typedef of *base\_func* and a rule prototype macro *base\_func\_def*. The defines are as follows:

```
typedef DWORD (_VMMAPIPTR base_func) (RPS *pRPS, WORD msg);

/* Base rule function prototype macro... */
#define base_func_def(fname) DWORD _VMMAPI fname(RPS *pRPS, WORD msg)
```

The RPS structure passed by address as a parameter is also defined in the RPLIB.H header file.

## Making a new base rule

When adding a base rule, you need to add it to the base rules list in CUSJDT.H and the custom rules array in CUSREG.H. The array lets you define a name by which you wish to identify the rule and the rule function that will be executed when that name is invoked. The registered name is the name you will use in the AFGJOB.JDT file. Usually the rule names are the same, or similar to, the actual function name, but this is not a requirement.

For example, in CUSJDT.H to add a function named *MyNewBaseFunc* you would add:

```
base_func_def(MyNewBaseFunc);
```

Then add the *MyNewBaseFunc* to the array in CUSREG.H:

```
/* String-to-Pointer array of custom rules... */
BaseFuncSym aBaseCustomRuleParms EMPTY() =
{
    { "BatchByPageCount", BatchByPageCount },
    { "ChubbCreateMacros", ChubbCreateMacros },
    { "CreateRecordList", CreateRecordList },
    { "CUSInitPageBatchedJob", CUSInitPageBatchedJob },
    { "InitPageBatchedJob", CUSInitPageBatchedJob },
    { "GetRCBRec", GetRCBRec },
    { "GetBatch", GetRCBRec },
    { "ImageRemover", ImageRemover },
    { "InitMerge", CUSInitMerge },
    { "InitPrint", CUSInitPrint },
    { "InstCustomConvertDateProcs", CUSInstCustomConvertDateProcs },
    { "InstCustomFormatNumberProc", CUSInstCustomFormatNumberProc },
    { "LoadFieldVars", LoadFieldVars },
    { "LoadGlobalFonts", LoadGlobalFonts },
    { "MyNewBaseFunc", MyNewBaseFunc },
    { "ProcessRcpRecs", ProcessRecord },
    { "ProcessRecord", ProcessRecord },
    { "PageBatchStage1InitTerm", PageBatchStage1InitTerm },
    { "PrintFormset", PrintFormset },
    { "PrintData", PrintData },
    { "MergeAFP", CUSMergeAFP },
    { "AddTextLabel", CUSAddTextLabel },
    { "AddLine", CUSAddLine },
    { "ParseCommentExample", ParseCommentExample },
    { "WriterCBWithPageCount", CUSWriterCBWithPageCount },
    { "CUSWriterCBWithPageCount", CUSWriterCBWithPageCount },
    #if defined(__OS22__) || defined(__WIN__) || defined(WIN32)
    { "CUSInitMail", CUSInitMail },
    { "MailFormset", MailFormset },
    #endif
    { "PrintFormsetListForDebugging", CUSPrintFormsetListForDebugging },
    { "PrintImageInfo", PrintImageInfo },
    { "WriteIniList", CUSWriteIniList },
    { "CUSInstallCommentLineCallback", CUSInstallCommentLineCallback },
    { "CUSHideImage", CUSHideImage },
    { "\0", NULL }
};
```



## Image Rules

These rules define actions to perform on single images within a form, based on a specific transaction. Image level rules typically affect the how an image is added to a given form. For instance, such rules might determine where the image will be placed (its origin) on a given form; what size it will have; and the propagation of field data once the image has completed processing.

Image level rules are stored in the Data Definition Table (DDT files).

### Prototypes

The image rule prototypes are found in the RPLIB.H file. There is a typedef of *image\_func* and a rule prototype macro *image\_func\_def*. The defines are as follows:

```
/* Image rule function definition...*/
typedef DWORD (_VMMAPIPTR image_func) (RPS *pRPS, WORD msg);

/* Image rule function prototype macro...*/
#define image_func_def(fname) DWORD _VMMAPI fname(RPS *pRPS, WORD msg)
```

The RPS structure passed by address as a parameter is also defined in the RPLIB.H header file.

### Making a new image rule

When adding an image rule, you need to add it to the image rules list in CUSJDT.H and the custom rules array in CUSFDT.H. The array lets you define a name to register the rule as and the rule function name. The registered name is the name you will use in the DDT file. Usually these names are the same as the actual function name.

For example, in CUSJDT.H to add a function named MyNewImageFunc you would add:

```
image_func_def (MyNewImageFunc);
```

Then add the MyNewImageFunc to the array in CUSFDT.H:

```
/* String-to-Pointer array of custom rules... */

ImageFuncSym aImageCustomRuleParms EMPTY() =
{
    { "AddImgAfterCurImg",          AddImgAfterCurImg},
    { "AddImgIfNextIsDiff",         AddImgIfNextIsDiff},
    { "AddImgIfNextIsSame",         AddImgIfNextIsSame},
    { "CUSMeterLines",              CUSMeterLines},
    { "CreateSubExtractList",        CreateSubExtractList},
    { "ImageTest",                  ImageTest},
    { "IncMacroValue",              IncMacroValue},
    { "MyNewImageFunc",             MyNewImageFunc},
    { "SetMacroValue",              SetMacroValue},
    { "SetCustChartAxisLabels",      SetCustChartAxisLabels},
    { "CusSetDynamicScaleAxis",      CusSetDynamicScaleAxis},
    { "\0",                          NULL}
};
```

## Field Rules

These rules define actions to perform on the variable fields in an image. Field level rules provide mapping, masking, and formatting information for each variable field on a form.

Field level rules are stored in the Data Definition Table (DDT files).

### Prototypes

The field rule prototypes are found in the RPLIB.H file. There is a typedef of *field\_func* and a rule prototype macro *field\_func\_def*. The defines are as follows:

```

/* Field rule function definition...*/
typedef DWORD (_VMMAPIPTR field_func) (RPS *pRPS, WORD msg);

/* Field rule function prototype macro...*/
#define field_func_def(fname) DWORD _VMMAPI fname(RPS *pRPS, WORD msg)

```

The RPS structure passed by address as a parameter is also defined in the RPLIB.H header file.

### *Making a new field rule*

When adding a field rule, you need to add it to the field rules list in CUSJDT.H and the custom rules array in CUSDDT.H. The array lets you define a name to register the rule as and the rule function name. The registered name is the name you will use in the DDT file. Usually these names are the same as the actual function name.

For example, in CUSJDT.H to add a function named MyNewFieldFunc you would add:

```
field_func_def (MyNewFieldFunc);
```

Then add the MyNewFieldFunc to the array in CUSDDT.H:

```

/* String-to-Pointer array of custom rules... */
FieldFuncSym aFieldCustomRuleParms EMPTY() =
{
    {"MoveBarcode",          MoveBarcode},
    {"MoveCityStateZip4",    MoveCityStateZip4},
    {"MoveDate",             MoveDate},
    {"MyNewFieldFunc",       MyNewFieldFunc},
    {"TextMerge",            TextMerge},
    {"CUSDisplayUsage",      CUSDisplayUsage},
    /* {"CUSmove_it",         CUSmove_it}, */
    {"CUSTableLookUp",       CUSTableLookUp},
    {"SelectRule",           SelectRule},
    {"\0",                   NULL}
};

```

## Recipient Rules

### *Prototypes*

The recipient rule prototypes are found in the RPLIB.H file. There is a typedef of *rcp\_func* and a rule prototype macro *recip\_func\_def*. The defines are as follows:

```

/* Recipient rule function definition...*/
typedef DWORD (_VMMAPIPTR rcp_func) (RPS *pRPS, RECIP_TBL_ENTRY
    *current_entry, int *RecsFound);

/* Recipient rule function prototype macro...*/
#define recip_func_def(fname) \
    DWORD _VMMAPI fname (RPS *pRPS, RECIP_TBL_ENTRY *current_entry, \
    int *RecsFound)

```

The RPS structure passed by address as a parameter is also defined in the RPLIB.H header file.

### *Making a new recipient rule*

When adding a recipient rule, you need to add it to the recipient rules list in CUSLIB.H and the custom rules array in CUSRCP.H. The array lets you define a name to register the rule as and the rule function name. Usually these names are the same as the actual function name.

For example, in CUSLIB.H to add a function named MyNewRecipFunc you would add:

```
recip_func_def (MyNewRecipFunc);
```

Then add the MyNewRecipFunc to the array in CUSRCP.H:

```

/* String-to-Pointer array of custom rules... */
RcpFuncSym aRcpCustomRuleParms EMPTY() =
{
    /* Insert Rules (see also prototypes in cuslib.h)...*/

    {"CUSnoopfunc",          CUSnoopfunc},
    {"MyNewRecipFunc",      MyNewRecipFunc},
    {"\0",                  NULL}
};

```

# Customizing Documaker Workstation

No program can be all things to all people therefore customizations are inevitable. The infrastructure of Documaker Workstation is designed in such a way that much of the functionality is handled through replaceable functions. This makes it possible to add, alter, or substitute functionality easily without having to change the base system. Another benefit of this replaceable functionality design is that it helps to ensure there is an upgrade path for customers with custom code. Isolating custom code from the base implementation allows new features to be added to the product without losing (or interfering) with the changes made for a specific customer's installation.

Documaker Workstation is designed to support several different types of external procedures. This document will cover the standard method of extending functionality by use of the remote access library, menu procedures, DAL procedures, and hook procedures. The purpose of this document is to reveal and define the external hooks provided in the libraries, not to explain or insinuate the way any hook should be used.

## Remote Access Library (RACLib and RacCo)

The Remote Access Library (*RACLib*) was created to give non-Docucorp applications the ability to start, stop, and control (to some degree) Documaker Workstation. The library provides API functions that can be called from any computer language that can interface to C functions in a DLL. In addition, an ActiveX component (*RacCo*) is provided for Windows NT/9x. Refer to the [Remote Access Library](#) document for details.

## Writing Custom Code

For more information on Documaker Workstation, refer to the Documaker Workstation [Supervisor Guide](#), the Documaker Workstation [User Guide](#), and the [DAL Reference](#). For information on the Docucorp API, refer to the online API documentation found with the release [\REL111\DOC\API\INDEX.HTM](#).

## CSTLIB

The CSTLIB library is where you should make code customizations for Documaker Workstation. This library is reserved for processing customizations and comes with sample functions. You can use and build upon these functions for customer installations.

## Defining Custom Functions

Most of the external functions that are recognized within Documaker Workstation are defined via INI options, although some are defined within the menu resource. These definitions are covered in detail later in this document.

Many of the function references that you can place in the INI or menu resource require a specific definition or syntax. However, all external definitions have one format option in common. The common element is the method used to identify a DLL name and an exported function name to call. This element is shown as:

DLL->FuncName

The DLL name comes first and the function name is separated by use of the “->” characters.

Case is important when defining the exported function name. Generally, the name should be typed in the same case manner that is used to define the function in the .DEF (export definition) file for the DLL. If the DLL is linked without case-sensitivity, it may be necessary to use all uppercase to define the function name. All Docucorp DLLs are linked with case sensitivity.

If the named function cannot be located in the specified DLL, a Docucorp style (and on some platforms and operating system) message box will appear revealing the error. In this event, the two most likely problems are the spelling of the function name in the INI file or the function was not included in the .DEF file for the DLL.

## Defining Custom Functions for Cross-Platforms

Most of the programs and DLLs provided by Docucorp are compiled and used on several different operating systems -- Windows 32-BIT (Windows 95, 98, 2000, and NT). With this in mind, Docucorp has designed the applications in a manner that allows the sharing of resources between these environments.

Because, it is sometimes desirable to have the executable programs for more than one operating system in the same location, Docucorp has unique OS specific names for its DLLs. For instance: VMOS2.DLL, VMMWIN.DLL, and VMMW32.DLL are all names of the “same” DLL compiled for different environments.

With this naming convention, it is necessary to support a method of defining external procedures using the *DLL->FuncName* method that can be used for all platforms without requiring separate INI or menu resources. This is accomplished via a name substitution group that can be included in the INI file.

Separate INI groups are recognized by WIN32 for defining substitute DLL names. These groups are named [ OS2SUBS ], [ WINDOWSUBS ] and [ WINDOW32SUBS ]. Since OS/2 was the original operating system for most Docucorp products, the OS/2 names are considered the key or primary name and, as such, should be used when defining external function references.

Consider the following example that identifies a hook procedure recognized by AFE.

```
[ AFEProcedures ]
PreEdit = TRNOS2->TRNPreEdit
[ WindowSubs ]
TRNOS2 = TRNWIN.DLL
[ Window32Subs ]
TRNOS2 = TRNW32.DLL
```

In this example, the first group identifies a hook procedure that is called in a specific situation. Since only one DLL name can be used in the definition, multiple platforms must be handled using the substitution group. In this example, alternate names are defined for Windows and for WIN32.

When a Docucorp program decides to load and query the specified hook, the DLL name is used as a search key within that operating system's substitution group. If an entry is located in that group, the newly found name is used instead of the one specified for the hook procedure. If an entry is not found, the original name will be used. In this example, OS/2 would use the name TRNOS2 because no [OS2SUBS] key was defined by that name; Windows would use the name TRNWIN.DLL taken from [WINDOWSUBS]; and WIN32 would use the name TRNW32.DLL taken from [WINDOW32SUBS].

---

All “base” DLL names, provided by Docucorp International, have been pre-registered in both the [ WINDOWSUBS ] and [ WINDOW32SUBS ] INI groups. The above example is used to merely demonstrate the type of INI changes required accessing DLLs not provided in a base release.

---

To help the Windows' environment to distinguish what module to load, it is often necessary to include the DLL extension on the names provided for substitution. All base DLLs are pre-registered in this fashion.

---

## MENU Procedures

A menu procedure is one of simplest methods of extending the AFE application or the DDS workbench. As the name implies, a menu procedure is called when the user selects a menu item.

### Menu Resource Format

Documaker Workstation reads an external file to create its menu. This file is identified by the INI option:

```
[MENU]
FILE = path\file name
```

Where *path\file name* specifies a menu resource file. The value associated with FILE should contain the file name (with a path specified if necessary) that identifies the external menu resource. By default, AFE INI files ship with a menu file named as MEN.RES or MENU.RES.

The DDS workbench reads an external file to create its menu. This file is located by retrieving an INI option from the FAPCOMP.INI file:

```
[MENU]
DDSFile = path\file name
```

The value associated with DDSFile should contain the file name (with a path if necessary) that identifies the external menu resource. By default, the file named as DDS.MNU is used.

Each line of the menu resource file begins with a *keyword*. A keyword determines how the rest of the menu resource line (statement) will be interpreted. All statement elements required to define a menu resource line are separated by a space. Extra white space and blank lines are ignored.

The definition of the menu resource file also contains security information. You can disable menu items for users who do not possess a sufficient security rating.

### Menu Keywords

#### *BEGIN and END*

Example:

```
POPUP "&File" 251 "System menu"
BEGIN
    MENUITEM "&New" 260 "AFEOS2->AFECreat" "New document" 9
    SEPARATOR
    MENUITEM "E&xit" 50000 "NULL" "Exit application"
END
```

These keywords do not have parameters and they must be used in matched pairs. All MENU and POPUP statements require a BEGIN and END to enclose the items that belong to those groups.

Old menu resource files may also contain the following obsolete keywords. PRODUCT1 was a second product name line. PRODUCT2 was generally used as a copyright notice. Moreover, POPUPBUTTON defined a menu level button for OS/2 versions.

---

## **BITMAP**

This keyword defines the bitmap that displays in the background of the window.

Syntax:

```
BITMAP bitmapname.bmp
```

Example:

```
BITMAP    collagem.bmp
```

## **BUTTON**

Each BUTTON statement defines a new button that is subordinate to the BUTTON or SUBBUTTON that contains the statement. This statement defines the text to appear and the function to call when selected.

Syntax:

```
BUTTON "item" ### "DLL->FuncName" "description" #
```

Examples:

```
BUTTON "Formset" "fdt.bmp" 4836 "FDTOS2->FDTEdit" "Run Formset Editor"
BUTTON "Image"    "img.bmp" 4837 "IMGOS2->IMGEEdit" "Run Image Editor"
```

"item" represents the text that will appear on the client area button.

The element ### represents a unique numerical value to associate with the menu item. Generally, client buttons are associated with menu items. If this is the case, then the menu item and function should match that put in your menu definition.

A standard definition used by Documaker Workstation to identify an external DLL and an exported function to call is expected as the next parameter. The string "NULL" can only be used if the function is an internally recognized menu item.

The DLL function indicator must conform to the format shown in the example. The DLL name comes first and the function name is separated by the "->" characters. Case is usually important when defining the exported function name.

A "description" parameter is next and may be "NULL" to indicate that no description is defined or necessary.

Security level is the last element of the BUTTON statement. The value may range from 0 to 9, where 0 is the highest (supervisor) level and 9 is the lowest (anybody) level. The security level will default to 9 if this parameter is omitted.

## **BUTTONS**

A BUTTONS statement must be followed by the BEGIN keyword. Each BUTTONS grouping must be closed with an END keyword. Only the first BUTTONS statement group is loaded from the resource file.

Example:

```
BUTTONS
BEGIN
    BUTTON "Formset" "fdt.bmp" 4836 "FDTOS2->FDTEdit" "Run Form Set Editor"
    BUTTON "Image"   "img.bmp" 4837 "IMGOS2->IMGEEdit" "Run Image Editor"
END
```

## **MENU**

This keyword defines the title of the menu or program and the title must be enclosed in quotes.

Syntax:

**MENU "program window title"**

Example:

```
MENU "Processing System"
```

A MENU statement must be followed by the BEGIN keyword. Each MENU grouping must be closed with an END keyword. Only the first MENU statement group is loaded from the resource file.

## **MENUITEM**

Each MENUITEM statement defines a new menu item that is subordinate to the menu or sub-menu that contains the statement. This statement defines the text to appear and the function to call when selected.

Syntax:

**MENUITEM "item" ### "DLL->FuncName" "description" #**

Examples:

```
MENUITEM "&New" 260 "AFEOS2->AFECreat" "New document" 9  
MENUITEM "E&xit" 50000 "NULL" "Exit application"
```

"item" represents the text that will appear on the parent MENU or POPUP that contains the statement. Ampersand (&) is used to indicate the "accelerator" letter for the menu item. This letter will be underlined by the operating system.

The element ### represents a unique numerical value to associate with the menu item.

A standard definition used by Documaker Workstation to identify an external DLL and an exported function to call is expected as the next parameter. The string "NULL" can only be used if the function is an internally recognized menu item.

The DLL function indicator must conform to the format shown in the example. The DLL name comes first and the function name is separated by the "->" characters. Case is usually important when defining the exported function name.

A "description" parameter is next and may be "NULL" to indicate that no description is defined or necessary.

Security level is the last element of the MENUITEM statement. The value may range from 0 to 9, where 0 is the highest (supervisor) level and 9 is the lowest (anybody) level. The security level will default to 9 if this parameter is omitted.

## **POPUP**

Each POPUP statement defines a new sub-menu that is subordinate to the menu or sub-menu that contains the statement.

Syntax:

**POPUP "item" ### "description"**

Example:

```
POPUP "&File" 251 "System menu"
```

The first string (enclosed in quotes) will be shown on the parent menu. The next element is a unique number to associate with the menu item. Finally, a short description of the menu item (enclosed in quotes) completes the line. The string "NULL" may be used for the description to indicate that no description is necessary or available.



The first string can include an ampersand (&) to indicate the following letter is the “accelerator” letter for the menu item. This letter will be underlined by the operating system.

Like MENU, the POPUP statement must be followed by a BEGIN keyword and is closed with an END keyword.

A POPUP statement that has the MENU statement as its parent is called a top-level popup. Top level popups will normally be seen on the program’s menu bar at all times.

Sub-level popup are permitted. This means that a POPUP statement may be included within another POPUP statement’s BEGIN and END grouping. It is possible to build a long descending chain of menu popups, if that is required.

---

If the all items contained within the BEGIN and END grouping are disabled due to security values, then the POPUP item will be disabled on its parent menu.

---

## SEPARATOR

No parameters are required on this statement. This keyword will cause a line to appear between the previous menu item and the next item read.

Example:

```
MENUITEM "&New" 260 "AFEOS2->AFECreat" "New document" 9
SEPARATOR
MENUITEM "E&xit" 50000 "NULL" "Exit application"
```

## SUBBUTTON

Each SUBBUTTON statement defines a new sub-menu that is subordinate to the BUTTONS or SUBBUTTON that contains the statement. A SUBBUTTON statement must be followed by the BEGIN keyword. Each SUBBUTTON grouping must be closed with an END keyword.

Syntax:

**SUBBUTTON “item” ### “description” #**

Examples:

```
SUBBUTTON "Resources" "res.bmp" 1 "Resource Programs"
BEGIN
  BUTTON "Fonts" "fxr.bmp" 4842 "FXROS2->FXREdit" "Run Font editor"
  BUTTON "Logos" "lgo.bmp" 4843 "LGOOS2->LGOEdit" "Run Logo editor"
END
```

The first string (enclosed in quotes) will be shown on the client area's push button. The next element is a unique number to associate with the menu item. Finally, a short description of the menu item (enclosed in quotes) completes the line. The string “NULL” may be used for the description to indicate that no description is necessary or available.

Like BUTTON, the SUBBUTTON statement must be followed by a BEGIN keyword and is closed with an END keyword.

SUBBUTTONs are very similar to POPUPs in menus. They cannot have a function associated with them, but when the user clicks on one, the set of buttons specified under it appears. Two functions are available for backing up in the hierarchy of buttons: FWMSHOWPrevButtons and FWMSHOWMainButtons, which back up one level and return to the first/main level, respectively. (See the example, above.)

The IDs assigned to buttons can be duplicates of menu items, or unique. If a button has a duplicate ID, the function associated with the menu item will be called when the button is pressed.

---

As a side effect of this enhancement, TOOLBAR items now have to go outside any BEGIN/END pair. (They originally went inside the BEGIN/END for the menu.)

For a program to fully support these buttons, it should have the following call in the WM\_SIZE case of its client window proc (This will re-center the buttons when the window is resized.):

```
FWMPositionButtons (hwnd, VMMNULLHANDLE);
```

For SUBBUTTONs, it's probably safest to assign unique Ids, not the same ID as the parallel menu item.

The buttons are about 20% of the window size, and approximately square.

Placement: You can use SEPARATOR to start a new row of buttons. Each row is centered horizontally, and the whole group of buttons is centered vertically.

---

## TOOL

The TOOL statement defines a button for a tool bar that will be displayed under the title bar. The TOOL statement items have to go outside any BEGIN/END pair.

Syntax:

TOOL bitmap offset, menu ID, button state, button type

Example:

```
TOOL 0 0 NULL SEPARATOR
TOOL 0 260 ENABLED BUTTON
TOOL 1 261 ENABLED BUTTON
TOOL 2 110 ENABLED BUTTON
```

## Menu Item IDs

Menu IDs are grouped into several categories.

ID Range	Description
100 - 200	Reserved for AFE procedures that are active while a form set is open but deactivated when no form set is open. The macro AFEISGFEID() will return TRUE or FALSE if a specified ID falls within this range.
100 - 150	A sub-category of the first range. These menu items are only activated while a form set is open that was not retrieved from an archive. The macro AFEISGFEARCID() will return TRUE or FALSE whether a specified ID falls within this range.
201 - 300	Reserved for AFE procedures that are active while a form set is not open but deactivated when a form set is open. The macro AFEISNONGFEID() will return TRUE or FALSE whether a specific ID falls within this range.
1000-1199	Reserved for GFE (the base Entry module) procedures that are active while a form set is open. These are disabled if a form set is not open. The macro GFEISGFEID() will return TRUE or FALSE whether a specific ID falls within this range. Since many of these IDs are automatically recognized by the base system without requiring the "DLL->Function" definition, do not add custom menu items in this range.

Any value not contained within these ranges is assumed to be active at all times. When assigning an ID to a new menu item, it will be important to determine which range to use or avoid. Each menu item must be unique.

## Menu Procedure Prototype

All menu procedures called by Documaker Workstation must conform to the following prototype.

```
int _VMMAPI funcname(HAB hab,
                     HWND mainhwnd,
                     VMMHANDLE menuH);
```

The parameters passed to the service function have the following meanings:

*HANDLE hab* is the program's anchor block or instance handle. The distinction depends upon whether the program is running on an OS/2 or Windows platform. Within the Docucorp programming environment, both definitions serve the same purpose.

*HWND mainhwnd* will be the window handle of the application that contains the menu.

*VMMHANDLE menuH* is the VMMHANDLE of the menu item's structure element. Each menu item has a structure definition created when the menu resource file is read. In general, it is not wise to manipulate the structure elements in the called function. Several macros/functions will return useful information from the structure, however the following two are especially useful within AFE.

```
PAFEDATA pdata = (PAFEDATA)FWMItemvPtr( menuH );
```

FWMItemvPtr() will return any pointer associated with the menu item. In AFE, this will return a pointer to the AFEDATA structure.

The AFEDATA structure is defined in AFELIB.H. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should be careful when you manipulate data within this structure.

```
ULONG id = FWMItemId( menuH );
```

FWMItemId() will return the unique identifier (ID) of the menu item. This ID can be used to enable or disable the menu item or used to determine its position within the menu.

In most cases, the return value of a menu procedure is ignored, but in general, it is a good idea to return whether the function succeeded (a zero value) or failed (a non-zero value).

## Menu Replacement

### *FWMLoadNewMenu function*

The FWMLoadNewMenu function that can be called from a menu resource to switch the menu. The name of the new menu file should be specified in the optional message area to the right of a menu-item definition.

Example:

```
MENUITEM "Conversion" 42 "GUIOS2->FWMLoadNewMenu" "conv.res"
```

## AFE Procedure Hooks

In programming terms, a *hook* is a method of gaining (or giving) control at or over a particular event during an application's execution. A *hook procedure* is a program function that is called by an *activated* hook.

A hook procedure is installed (and the hook activated) by assigning a value to a hook option in the INI file. Each hook provided in AFE is designed to give access to a specific situation.

Although hooks are registered via the INI file, not all hooks procedures have the same programming requirements. In many cases, different hooks require entirely different procedure prototypes. Additionally, some are expected to return certain values to indicate success or failure.

## INI Options

Most hooks within the AFE Entry module are defined via INI options. Although some hooks require specific definitions that differ from others, all hooks registered by INI have one format option in common. The common element is the **DLL->FunctionName** reference to identify a DLL name and an exported function name to call.

Since each hook does not expect the same INI option syntax, the specific registration requirements will be included in the hook description provided in this document.

## Hook Prototypes

Not all hooks require the same function prototype; however, most use one of several basic prototypes. Within the hook descriptions provided in this document, the prototype requirements will be identified. If a special prototype is used, it will be defined in the hook description.

### FAPUSER

This prototype definition can be found in the header file **fapuser.h**. The prototype takes the following form.

```
typedef FAPDW (FAPAPIPTR FAPUSERPROC) (FAPDW dwMessage,
                                         FAPDW dwFAPHab,
                                         FAPDW dwFAPHwnd,
                                         FAPDW dwObjectIdentifier,
                                         FAPDW dwObjectType,
                                         FAPDW dwInputFlag1,
                                         FAPDW dwInputFlag2,
                                         FAPDW dwInputFlag3,
                                         char FAR * lpszObjectName,
                                         char FAR * lpszFormatType,
                                         char FAR * lpszFormat,
                                         char FAR * lpszEditData,
                                         char FAR * lpszInputBuffer,
                                         char FAR * lpszOutputBuffer,
                                         FAPDW dwOutputBufferMaxSize,
                                         FAPDW FAR * lpdwOutputFlag1,
                                         FAPDW FAR * lpdwOutputFlag2,
                                         FAPDW FAR * lpdwOutputFlag3);
```

Since this is a common prototype, it is also generic. The large number of parameters evolved over time to ensure that enough basic information could be passed to each function to perform its task. Since the prototype serves as a generic definition, the specific value referenced (or contained) by each parameter may differ with each hook procedure. In fact, it is common for parameters to be omitted or unavailable, indicated by values set to NULL or zero.

The description of each hook procedure that uses the FAPUSER prototype will include the definition and expected use of the parameters that the hook supplies.

*TSTOS2->UserTest* is a test function exported from the specified DLL that conforms to the FAPUSER prototype. In many cases, this function may be used to test that a hook is activating properly by displaying a window containing some of the information passed to the function. However, please note that this function has no idea what it should do with any messages passed to it and always return SUCCESS.

### FAPHANDLER

Generally, this prototype is used by hook functions related to Docucorp object message handling. Exceptions to this rule will be noted in the specific hook description. This prototype is defined in FAPFORM.H and takes the following form:

```
typedef DWORD (_VMMAPIPTR FAPHANDLER)(VMMHANDLE objectH,
                                       DWORD msgno,
                                       FAPPARM p1,
                                       FAPPARM p2);
```

The *objectH* parameter usually represents a VMMHANDLE to a FAPOBJECT. Each FAPOBJECT (defined in FAPFORM.H and structures defined in FAPDEF.H) has a registered message handler that acts similar to the way window procedures handle messages for a window. In some cases, hooks are used to replace these handlers overriding the default functionality and in other cases, only subsets of an object's messages are passed to a specific hook procedure.

The *msgno* parameter contains the specific message number being passed to the function. Each type of message used by FAPOBJECTs must be unique. There is a list of pre-defined messages in FAPFORM.H, but this list may be extended by defining your own messages with FAP\_MSGUSER + *n*, where *n* represents some number greater than zero.

The remaining parameters may contain values to be used by the functions and are specific to the message number being passed. The variable type FAPPARM is a redefinition of the union FSIPARM that contains several variable types that might be received. FSIPARM is defined in FSL.H and takes the following form:

```
typedef FSIPARMTYPE _FSIPARM
{
    VMMHANDLE vmh;
    VOID FAR *ptr;
    FAPPFN    fn;
    DWORD     dw;
    WORD      w;
    BYTE      b;
} FSIPARM;
```

Included in the union definition are references for VMMHANDLES and FAPPFNs. A FAPPFN is a pointer to a void function. If the value represents some other function prototype, an appropriate cast will be required to call the procedure or assign it to another variable.

## INI Settings

```
[ AfeProcedures ]
AFERetDisplLstHook = AFEOS2->AFERetDisplayList
AFERetriB4AppendgToLstHook = DLL->FunctionName
AFERetriOkButtonHook = DLL->FunctionName
Archive = AFEOS2->AFEWip2ArchiveRecord
Archive2WIP = AFEOS2->AFEArchive2WipKeys
AutoKeyID = TRNOS2->TRNAutoKeyIDUsrFunc
BannerProc = TRNOS2->TRNSetBannerFormInfo
BUTTON1 = TRNOS2->TRNAutoNextKey
BUTTON2 = AFEOS2->AFEPersonalEdit
BUTTON3 = TSTOS2->UserTest
CheckUserEntry = DLL->FuncName
Complete = DLL->FuncName
EntryFormset = DLL->FuncName
IndexName = CUSOS2->CUSGetArcIdxName
Init = DLL->FuncName
Parse = DLL->FuncName
PostEdit = DLL->FuncName
PreEdit = DLL->FuncName
Term = DLL->FuncName
WindowProc = DLL->FuncName
Security = AFEOS2->AFESecurityFunc
Wip2Archive = AFEOS2->AFEWip2ArchiveKeys
```

## Functions and Hooks

Function/Hook	Result
<a href="#">AFE Append to Record Hook</a>	This hook will be called immediately before a new record is added to the static list, after the OK button on the Retrieve window is pressed.
<a href="#">AFE Archive List Hook</a>	This function is primarily responsible for retrieving records from the archive database and adding them to a static list that will be displayed for archive retrieval.
<a href="#">AFE Archive Record Selected Hook</a>	This hook will be called immediately after the OK button from the Retrieve window is pressed.
<a href="#">AFE Check Form Set Data Hook</a>	This hook is called before a new WIP entry can be created to allow the function to check form set data.
<a href="#">AFE Complete Form Set Hook</a>	This hook is called at three points in the Complete action taken by the user thus allowing customized features to be added to Complete.
<a href="#">AFE Entry Form Set Hook</a>	A form set appears on the user's screen and a series of functions are performed. For instance, the form set is filtered by removing any forms that are not selected and Required forms are checked for inclusion. This hook was created to allow customization within this process.
<a href="#">AFE Form Selection Buttons Hook</a>	Buttons on Form Selection window.
<a href="#">AFE Initialization Hook</a>	This hook is called any time the INI settings are loaded. Usually this occurs when the program starts and when master resources are changed.
<a href="#">AFE Parse Command Line Hook</a>	This hook is called to allow the command line parameters to be parsed and used in custom code rather than base code.
<a href="#">AFE Post Edit Hook</a>	This hook is called before a form set is saved (unloaded), assigned to another user, completed, or deleted.
<a href="#">AFE Pre Edit Hook</a>	This hook is called after a form set has been loaded successfully and before the Entry module can begin.
<a href="#">AFE Termination Hook</a>	This hook is called when the Entry module is exiting.
<a href="#">AFE Window Procedure Hook</a>	This hook is designed to let a custom function intercept messages that arrive at the main application's window procedure.
<a href="#">AFEArchive2WipKeys</a>	Translates the archive Key1, Key2, and KeyID fields into the corresponding WIP fields.
<a href="#">AFESecurityFunc</a>	Verifies a user has access rights to enter the program
<a href="#">AFEWip2Archive</a>	Translates the WIP Key1, Key2, and KeyID fields into the corresponding Archive fields.
<a href="#">AFEWip2ArchiveRecord</a>	Creates the archive index record from a WIP record.
<a href="#">AppldxRec</a>  Use this function to get an archive record based on APPIDX.DFD and Trigger2Archive INI settings.  <b>Syntax:</b> AppldxRec ()  <b>Example:</b> Comment = AppldxRec() AddComment( Comment ) CUSGetArcldxName	Use this function to get an archive record based on APPIDX.DFD and Trigger2Archive INI settings.

<a href="#">CUSGetArclIdxName</a>	Obtains the archive index name of a specific WIP record.
<a href="#">TRNAutoKeyIDUshrFunc</a>	Auto-Fill Key ID on Form Selection window.
<a href="#">TRNSetBannerFormInfo</a>	Set banner page information.

## Transactions

Form sets may be created with different requirements. One way of delineating the purpose of a form set is by identifying it with a Transaction Type. Several different transaction types are supported by the base system including, “New Business”, “Renewal”, “Quote” and “Endorsement”. Each of these transaction types will change (or specify) certain criteria about the form set selection process. The Documaker Workstation [Supervisor Guide](#) details customizing WIP transaction code and for more specific information on these base functions, refer the TRNLIB section of the [API Documentation](#).

### INI Definition

Any number of transaction types may be defined by the user. Each may have a special custom function or they may share the same custom function. The following example demonstrates two transaction registrations.

```
[ Transactions ]
01 = ;NB;New Business;TRNOS2->TRNNew;
02 = ;QU;Quote;TRNOS2->TRNNew;
```

The left side of the equation is used only to delineate each item. The right side of the definition includes the transaction code (2 characters); followed by the transaction type name; followed by the standard definition for DLL and function name.

### Syntax

Transaction functions must conform to the FAPHANDLER prototype. For more information on the FAPHANDLER prototype, see the [FAPHANDLER](#) section. You can use the following parameters:

Parameter	Description
<i>ObjectH</i>	The handle of the current form set.
<i>Msgno</i>	A requested operation message number. Currently, only the FAP_MSGINIT message is sent.
p1	Is a union. p1.ptr will be NULL or contain a pointer to a BOOL variable type. If a valid pointer is passed, the custom function should set the BOOL to TRUE if the form set should enforce a unique KeyID, otherwise it should be set to FALSE. (Some applications allow duplicate keys on certain transaction types.)
p2	Is a union. p2.ptr is a pointer the TRANSREC structure associated with the form set. This structure is defined in AFELIB.H. By examining this structure for the transaction name, it is possible to have several transactions share the same custom function.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

A custom transaction function should use care when manipulating the form set passed as a parameter. In most cases, the user is working with the Form Selection window when this hook is called.

# DAL Functions and Procedures

The Document Automation Language (DAL) is a scripting language that enhances form data collection during form entry or during the execution of the Docucorp Batch Processing. Through DAL, it is possible to access and change the values contained within form fields. A DAL calculation may perform mathematical operations, call functions, create variables, or even call other DAL calculations.

DAL is also extensible. Through installable script language functions and procedures, it is possible to create new or replace existing DAL functions and procedures.

This document will explain some of the steps necessary to install DAL procedures, but for a complete explanation about programming for DAL, you should refer to the [DAL Reference](#).

## INI Registration

It is possible to create and register additional procedures and functions for DAL. Once registered, DAL will automatically call a routine each time a script reference to the routine's name is encountered. It is actually possible to override an existing built-in routine by registering another one with the same name.

Although there is an internal API that can be called to register DAL routines, INI registration is also supported by the AFE Entry module.

```
[ DALFunctions ]  
KeyWord   = DLL->FuncName1  
KeyWord2  = DLL->FuncName2
```

The example above demonstrates INI registration of two DAL functions. Any number of functions can be registered in this manner.

The value to the left of the equal sign represents the DAL “verb”, or keyword, that will be used in DAL scripts when the specified function should be called. Valid DAL keywords may not contain a space. Some special symbols are permitted but not mathematical symbols or parentheses. Case is not important on DAL keywords.

After the equal sign, the standard DLL->FuncName convention is used to name a DLL to load and an exported function to call. Remember that case is usually important when naming DLL exported functions.

The routines named in these INI options are registered after the internal “built-in” functions. When a keyword is registered a second time, the latter registration will override the first. This makes it possible to substitute your own functionality for internally defined DAL functions and procedures.

## DAL Function Prototype

There are very few requirements for creating built-in functions or procedures. First, a routine must conform to this prototype:

```
DALERR_CODE _VMMAPI function(DALMODETYPE mode);
```

This prototype defines the procedure as being exported and of type \_VMMAPI. The return type, DALERR\_CODE, is a long value that will represent one of the internal error numbers.

The parameter, DALMODETYPE, is an enumerated value that identifies how the routine was called. A value of DALMODE\_FUNCTION indicates a return value is expected and DALMODE\_PROCEDURE means no return value is expected. Generally, a routine is created to be either a function or procedure. A function returns a value, while a procedure does not. The language syntax requires that values returned by functions be used as parameters to other routines or acted upon by operators. Conversely, procedures must be “stand-alone” statements and cannot be used as function parameters or other expressions. This distinction is necessary to help prevent possible errors by users (Re. script writers).

It is possible to write a routine that can serve as both a function and procedure. In this event, the *mode* parameter should be checked near the end of the routine to determine whether to push a result onto the DAL stack. If called as a function, this result is returned to be used in the calling expression.



A routine should return DALERR\_SUCCESS to indicate a successful completion. Any other value is interpreted as the error number to send to the currently registered error handler.

---

Specific requirements and information on building DAL functions and procedures is covered in a separate document detailing the [DAL Reference](#).

---

## Edit Functions

By assigning edit functions to a variable field, you can have the system execute specific functions before (pre-edit) or after (post-edit) a data entry user enters data into the field or both.

### Prototypes

The edit function must follow the *FAPUSERPROC* prototype can be found in the FAPUSER.H file. The *FAPUSERPROC* is a generic function prototype used by the system. The following function prototype is true for edit functions:

```
FAPDW FAPAPI EditProc(FAPDW msg,           // FAP_MSGPREEDIT/FAP_MSGPOSTEDIT
                      FAPDW dwFAPHab,      // Anchor block
                      FAPDW dwFAPHwnd,     /* Window handle of client
                      FAPDW fieldH,        /* Field handle */
                      FAPDW FAObjType,     /* Value is FAP_OBJFIELD
                      FAPDW flag,          /* FAPFIELD.flag = FFLAG_* values
                                           defined in FAPFORM.H. */
                      FAPDW Required,      /* FAPFIELD.required */
                      FAPDW Scope,         /* FAPFIELD.scope =
                                           SCOPE_LOCAL_IMAGE
                                           SCOPE_GLOBAL_FORM
                                           SCOPE_GLOBAL_FORMSET */
                      char FAR * Name,     /* FAPFIELD.name */
                      char FAR * FEType,   /* FAPFIELD.fetype[0] =
                                           C Custom
                                           x Alphanumeric
                                           k Int'l Alphanumeric
                                           a Alphabetic
                                           i Int'l Alphabetic
                                           X Uppercase Alphanumeric
                                           K Int'l Uppercase Alphanumeric
                                           A Uppercase Alphabetic
                                           I Int'l Uppercase Alphabetic
                                           n Numeric
                                           y (Y)es or (N)o
                                           m X or space
                                           d Date format
                                           t Table only
                                           M Multi-line text
                                           B Bar code
                                           T Time format
                                           &FAPFIELD.fetype[1] =
                                           language. If null then
                                           neutral. Otherwise is a
                                           UTL_LOCALE_* value defined
                                           in UTLFMT.H. */
                      char FAR * Format,    /* FAPFIELD.format */
                      char FAR * EditData, /* Data defined by your custom
                                           edit function. This value is
                                           supplied by the user at form
```

```

composition time on the field
properties edit tab's data
prompt. */
char FAR * inBuf,      // PPS Entry buffer (same as out)
char FAR * outBuf,     // PPS Entry buffer (same as in)
FAPDW outBufSize,      // PPS Entry buffer size
FAPDW FAR * outFlg1,   // Not used
FAPDW FAR * outFlg2,   // Not used
FAPDW FAR * outFlg3); // Not used

```

## Pre-Edit Functions

When you assign a pre-edit function to a variable field, the system executes that function before the user enters new information in the field. For example, you can assign a pre-edit procedure that inserts default information into the field. For a pre-edit function, the message sent is `FAP_MSGPREEDIT`.

## Post-Edit Functions

The system applies post-edit functions after the data entry user finishes entering data in the field. For example, you might assign a post-edit procedure to tell the system to create a cover letter using the name and address entered into the variable field. For a post-edit function, the message sent is `FAP_MSGPOSTEDIT`.

## Image Functions

By assigning image functions to an image, you can have the system execute specific functions when an image is opened, closed, or both.

## Prototypes

The image function must follow the *FAPUSERPROC* prototype that can be found in the `FAPUSER.H` file. The *FAPUSERPROC* is a generic function prototype used by the system. The following function prototype is true for image functions:

```

FAPDW FAPAPI ImageProc(FAPDW msg,      // FAP_MSGOPEN/FAP_MSGCLOSE
                        FAPDW dwFAPHab, // Anchor block
                        FAPDW dwFAPHwnd, // Window handle of client
                        FAPDW imageH,    // Image handle
                        FAPDW FAPObjType, // Value is FAP_OBJIMAGE
                        FAPDW dwInputFlag1, // Not used
                        FAPDW dwInputFlag2, // Not used
                        FAPDW dwInputFlag3, // Not used
                        char FAR * Name,    // IMAGENAME(imageH)
                        char FAR * FEType, // Not used
                        char FAR * Format,  // Not used
                        char FAR * EditData, /* Data defined by your custom
                                              image function. This value is
                                              supplied by the user at form
                                              composition time on the image
                                              properties edit tab's data
                                              prompt. */
                        char FAR * inBuf,   // Not used
                        char FAR * outBuf,  // Not used
                        FAPDW outBufSize,   // Not used
                        FAPDW FAR * outFlg1, // Not used
                        FAPDW FAR * outFlg2, // Not used
                        FAPDW FAR * outFlg3); // Not used

```

## Open Functions

When you assign an open function to an image, the system executes that function when that image gets a FAP\_MSGOPEN message. The image will broadcast the FAP\_MSGOPEN to all of its children then call the custom open function. For example, you can assign an open procedure that inserts default information into several fields on an image.

## Close Functions

When you assign a close function to an image, the system executes that function when that image gets a FAP\_MSGCLOSE message. The image will call the custom close function then broadcast the FAP\_MSGCLOSE to all of its children.

## Export Formats

A list of available export formats can be defined in the INI file. If more than one method is supported, the program will prompt the user to select the method that should be used.

No default values are assumed, however the standard function, TRNExport(), may be activated in the base system.

### INI Definition

Any number of export methods may be defined by the user.

```
[ ExportFormats ]
01 = ;xx;Export;TRNOS2->TRNExport;
```

The left side of the equation is used only to delineate each item. The first place holder on the right side of the definition not used. The second is the export method name and is followed by the standard definition for DLL and function name.

### Syntax

Transaction functions must conform to the FAPHANDLER prototype. For more information on the FAPHANDLER prototype, see the [FAPHANDLER](#) section. You can use the following parameters:

Parameter	Description
<i>ObjectH</i>	The handle of the current form set.
<i>msgno</i>	A requested operation message number. Currently, only the FAP_MSGINIT message is sent.
<i>p1</i>	Is a union. p1.ptr will be a pointer the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
<i>p2</i>	Is a union. p2.dw is a HWND value of the window that initiated the call.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

A custom transaction function should use care when manipulating the form set passed as a parameter. In most cases, the user is engaged with the Complete Form Set window when this hook is called.

## Import Formats

A list of available import formats can be defined in the INI file. If more than one import method is supported, the program will prompt the user to select the import method that should be used.

No default values are assumed, however two standard functions may be activated in the base system: TRNImport() and TRNSelImport().

### INI Definition

Any number of import methods may be defined by the user. The following example demonstrates two import method registrations.

```
[ ImportFormats ]
01 = ;xx;Standard;TRNOS2->TRNImport;
02 = ;yy;Selective;TRNOS2->TRNSelImport;
```

The left side of the equation is used only to delineate each item. The first place holder on the right side of the definition not used. The second is the import method name and is followed by the standard definition for DLL and function name.

### Syntax

Transaction functions must conform to the FAPHANDLER prototype. For more information on the FAPHANDLER prototype, see the [FAPHANDLER](#) section. You can use the following parameters:

Parameter	Description
<i>ObjectH</i>	The handle of the current form set.
<i>msgno</i>	A requested operation message number. Currently, only the FAP_MSGINIT message is sent.
<i>p1</i>	Is a union. p1.ptr will be a pointer the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
<i>p2</i>	Is a union. p2.dw is a HWND value of the window that initiated the call.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

A custom transaction function should use care when manipulating the form set passed as a parameter. In most cases, the user is engaged with the Form Selection window when this hook is called.

## Document Set Procedures

The documents set files read and write code can be replaced through INI options that specify what function to call when certain operations are requested.

Documents set files include the NA, POL and PKG files. Default functionality is provided for reading and writing these files as ordinarily DOS files if another INI option is not provided. These functions can be overridden to achieve an alternate method of reading and writing this information.

## INI Settings

Two separate group definitions are used to distinguish between document files in archive mode and non-archive mode. This was designed in a manner to assist those that might require their archived information to be retrieved from another location or in different manner than WIP files. The WIP group is [ AFEDSProcedures ] while the archive group is [ AFEDSArchiveProcedures ].

```
[ AFEDSProcedures ]
APPEND = DSOS2->DSDefAppendBuffer
CLOSE = DSOS2->DSDefCloseBuffer
CREATE = DSOS2->DSDefCreateBuffer
FIRST = DSOS2->DSDefFirstBuffer
NEXT = DSOS2->DSDefNextBuffer
OPEN = DSOS2->DSDefOpenBuffer

[ AFEDSArchiveProcedures ]
APPEND = DSOS2->DSDefAppendBuffer
CLOSE = DSOS2->DSDefCloseBuffer
CREATE = DSOS2->DSDefCreateBuffer
FIRST = DSOS2->DSDefFirstBuffer
NEXT = DSOS2->DSDefNextBuffer
OPEN = DSOS2->DSDefOpenBuffer
```

## Functions

Function	Result
<a href="#">DSDefAppendBuffer</a>	Appends a buffer of data to the current file.
<a href="#">DSDefCloseBuffer</a>	Closes the current file.
<a href="#">DSDefCreateBuffer</a>	This hook is called to create a new or truncate an existing file for writing.
<a href="#">DSDefFirstBuffer</a>	Return the first buffer of data from the current file.
<a href="#">DSDefNextBuffer</a>	Return the next buffer of data from the current file.
<a href="#">DSDefOpenBuffer</a>	Opens an existing file.

## Timed Service Functions

TMRLIB (Timer library) is a base library used to register and call service functions at specified time intervals. The Timer library was designed to work with most of the desktop applications created by Docucorp, especially Documaker Workstation. This document will specifically focus on the use of TMRLIB within Documaker Workstation while identifying areas that may differ when used by other programs.

AFEMAIN is the starting point for most Processing System programs. The AFEDATA structure, used by AFELIB, is supplied as a parameter to any timed functions registered by this program. Control over many aspects of the program's environment is available, including any currently selected form set and WIP record, with access to the current AFEDATA structure.

Other programs may use TMRLIB that do not use an AFEDATA structure. These programs can register an application specific pointer (to any type of data ) that will be passed to registered timed functions. In theory, these functions will know what that data pointer references and how to use it.

Several Docucorp libraries are referenced by TMRLIB, but the interaction with these is designed to be limited preventing dependence upon any particular version of our products. Although TMRLIB was not created until version 9.0, it should be possible to use TMRLIB in prior versions of the software (recompiling may be necessary). Registered service functions should do their own version checking of system libraries or other libraries when a specific functionality is required.

## History

On occasion, the Professional Services Group has been required to provide an interface between the Documaker Workstation system and an external program/event. Usually, this interface involved writing a daemon program to periodically check for these “events” and react when they were detected. The daemon programs either used an operating system timer or simply polled (in a loop) waiting for a specific event to occur.

Historically, Docucorp daemon programs start Documaker Workstation with command line parameters identifying what must be done. Once the task completes, the user must exit Documaker Workstation to reactivate the daemon program that then waits for another event.

Implementing a solution using this method did not lend itself to base support. To become a useful base feature, we needed to remove the requirement for the daemon program and simply make Documaker Workstation wait for the necessary event to occur. This eliminates starting and exiting the program repeatedly which is extremely time consuming. In some cases, program startup delay might allow other workstations waiting to act on the same event (such as a file appearing in an import subdirectory) to conflict over the task.

Another objection to the daemon program method is that it usually requires making changes to Documaker Workstation (AFEMAIN.EXE). Typically, these changes involved customizing the parameter list handling to call the necessary operations. By changing a base program in this manner, users cannot readily upgrade when newer version becomes available.

Finally, if the event or task that requires a response must occur within Documaker Workstation itself, this daemon program approach cannot easily provide a solution.

## Goals

The primary goal of TMRLIB is to make it possible to implement the functionality of the daemon program without changing AFEMAIN.EXE or even writing that third program.

The skeletal framework is built around “timed” calls to service functions rather than simple polling. This will allow the program to continue to run and check for required events as a background operation.

Since base support cannot anticipate what type or number of events each customer might require, these service functions will be registered via entries in the INI file. This should not require any modifications to base libraries or programs under normal circumstances.

In addition, a filtering capability is included to allow the registration of a service function to indicate what “state” the program should be in before being called. *State* will encompass whether a form set is loaded or not and whether the user is engaged in open windows or menus.

Finally, this implementation should make it possible to maintain the customer’s upgrade path with minimal effort.

## *Timed vs. Timer*

To avoid confusion, those functions called by this library will be referred to as “service” functions -- not timers.

True timers are a limited resource on most operating systems (OS). Because of this fact, all service functions that are registered and called by this library will share a single timer. Multiple service functions sharing the same time interval requirement will be called in the sequence order that they occur when retrieving INI file options.

## INI Settings

Timed service functions must be registered. Registration is accomplished by adding lines to an INI group that conforms to the following prototype.

```
[TIMERFUNCS]
REF = ;STATE;URGENCY;SECONDS;DLLNAME->FuncName;\DATA
```

The semicolons (;) are a required part of each registration statement. The DATA element is optional. If you include it, precede it with a backslash (\).

## REF

**REF** is simply a placeholder to distinguish each entry under the INI group, TIMERFUNCS. Each registered function should have a different REF value. The actual value is not used by TMRLIB. In most cases, you may wish to use simple ASCII numbering, such as 01=, 02=, 03=, and so on, to distinguish each service function line.

---

Although the **REF** value is not used by TMRLIB, remember that INI files are sorted when loaded. If the sequence of the service functions is important, the REF values should be established in a manner that will not be changed when sorted.

---

## STATE

The first Placeholder **STATE** is a *mode* or *state of program* flag. The STATE (in combination with URGENCY) indicates at what point during processing it is valid to call the service function. This indicator only applies to service function calls triggered by the timer. Initialization and termination affects all registered service functions regardless of the setting indicated on the registration line.

This valid values for this flag and their meanings are:

- 0= Desktop closed (No form set is currently loaded or in view)
- 1= Desktop open (A form set is currently loaded and in view)
- 2= Call any time (Use with caution)

The desktop is considered “opened” when a form set is currently being viewed and/or entry is active, otherwise the desktop is considered “closed”.

---

The desktop is considered opened when any form contained within the current form set, (retrieved by FAPFormset) has a FAPWINDOW associated with it.

---

If a service function should only be called when the desktop is closed, assign ‘0’ as the STATE. Note however, even if the desktop is closed, service function registered at STATE level 0 will not be called if the URGENCY requirement is not satisfied. In Documaker Workstation, STATE level 0 might be used to implement features like automatic import, or automatic WIP edit.

Use STATE level 1 when a service function should only be called while the desktop is opened. (Remember, this means that a FAPWINDOW is associated with a form in the current form set.) Note however, service functions are not called if the URGENCY requirement is not satisfied. In Documaker Workstation, STATE level 1 might be used to implement a feature like auto-save to WIP.

STATE level 2 should be used with discretion. Any service function using STATE level 2 will be called whether the desktop is opened or closed as long as the URGENCY requirement is satisfied. Service functions registered as STATE level 2 should probably not attempt to alter current forms or change the form set management. Doing so might cause the program to crash and burn.

Since STATE level 2 is active whether the desktop is opened or closed, this setting should only be used to implement features that do not hinder user operations and do not rely upon form set management.

## URGENCY

Once the STATE setting has been satisfied, the **URGENCY** flag will be evaluated. URGENCY represents how “timely” the call to the service function should be enforced. This setting may skip or delay the call if the user is engaged with an open window or menu. This indicator only applies to service function calls triggered by the timer. Initialization and termination calls all registered service functions regardless of the setting indicated on the registration line.

This valid values for this flag and their meanings are:

- 0 = Not Urgent (okay to bypass if window or menu is open)
- 1 = Rush (call as soon as possible after window or menu is not open)
- 2 = Urgent (call even if window or menu is open – use with caution)

---

TMRLIB subclasses the main window of the application and looks for messages that indicate the menu is active. In addition, all child windows associated with the application window are scanned to determine if any of them are windows. These two tasks are used to determine when URGENCY should be enforced.

---

A setting of zero (0) indicates that the call to the service function should be skipped if a window or menu controlled by the program is open. Skipping the call means that the service function will not activate again until the registered time interval's next elapses.

URGENCY setting equal to one (1) will call the service function if no window or menu is active. However, this setting will delay the call, rather than skip it, if a window or menu is open.

Delayed service functions are evaluated approximately every second (based upon the system timer) to determine when the call can safely go through. Delaying the call means the interval between when the call finally goes through and the next standard interval registered with the function may be reduced.

Also note, however, that time interval is not accumulated. If the delay causes the function to miss two or three time intervals, it will only be called once when the URGENCY is finally satisfied.

Use URGENCY level 2 with discretion. This setting will call the service function without regard for whether a window or menu is active. Service functions registered with an URGENCY level 2 should probably not attempt to alter current forms or change the form set management. Doing so might cause the program to crash and burn.

Since the user might be interacting with a window, this setting should only be used to implement features that do not affect or hinder user operations. In addition, these functions should probably not open windows. Imagine the user's frustration, if while completing the Print window, the program suddenly switches to another task or window.

## SECONDS

The **SECONDS** placeholder is a time-out value that designates (in seconds) how often the service function should be called. SECONDS can be any value from 1 to 32767. (The maximum value exceeds nine hours). Any line that contains a SECONDS value equal to zero (0) will be skipped.

Although, the time-out value is designated in seconds, the actual time is only approximated. This is covered in more detail later in this document.



Internally, two “time” values are maintained. One of these values is used exclusively to test when STATE level 2 functions should be called. Remember, STATE level 2 functions are called whether the desktop is opened or closed.

---

STATE levels 0 and 1 use a second time value maintained by TMRLIB. Only those functions satisfied by the current STATE flag (desktop opened or closed) will be called when the proper time interval has elapsed. When the desktop is opened or closed, the second time value is reset to zero. This guarantees that the time interval associated with a function must elapse before being called when the desktop state changes.

---

Calling a service function too frequently may slow program performance.

---

### *DLLNAME->FuncName*

The placeholder, **DLLNAME->FuncName**, is the standard FSI method used to identify a DLL to load and an exported function to call. **DLLNAME** should be a valid DLL name and **FuncName** much match a name that can be “queried” from that DLL. Only functions named in a DLLs **export** list can be referenced by TMRLIB.

As mentioned previously, the function identified by this option must conform to the FAPHANDLER function prototype.

### *DATA*

This registration member is optional. **DATA** should only be declared if the service function requires it. There is no format requirements established for the data line other than it must begin with a backslash (\).

Function specific DATA is attached, as a string of ASCII characters, to the FSITIMERREC structure associated with the registration line. It is the service functions responsibility to verify the existence or validity of the line.

---

The leading backslash will not appear in the data member of the structure.

---

## Example Registrations

Theoretically, there is no limit to the number of functions that you can register. Each can have it's own values for each of the registration parameters.

---

The more service functions that are registered, the greater the possibility that program performance will be adversely affected.

---

```
[TIMERFUNCS]
A=;0;0;60;TMROS2->TMRTimerTest1;
02=;1;1;30;TMROS2->TMRTimerTest2;
CHECK=;2;2;300;TMROS2->TMRTimerTest3;
```

This example registers three timed service functions. None of them has specified any function specific data to associate with the service function record.

TMRTimerTest1 will be called *approximately* every 60 seconds while the desktop is closed and the user is not engaged in a window or menu selection. If a window or menu is open at the timed interval, the function will be skipped until the next 60-second interval elapses.

TMRTimerTest2 will be called *approximately* every 30 seconds while the desktop is open and the user is not engaged in a window or menu selection. However, if a window or menu is open, the function will be delayed (not skipped) until the window or menu closes.

*Approximately* each second skipped functions are re-evaluated. When the program is satisfied that a window or menu is no longer open, the call will be made, regardless of whether the current time matches the registered interval. If multiple intervals elapse during the delay, only one call will be made to the function. Once a successful call has been made, the testing for this function's time interval returns to normal.

TMRTimerTest3 will be called *approximately* every five minutes whether the desktop is opened or closed and without regard to whether the user is engaged in a window.

As mentioned in the discussion of the last topic, the semicolons (;) are essential to distinguish the parts of the registration line. If a line cannot be parsed correctly, an error message will be displayed and the line skipped.

---

Each of these registration lines has a different **REF** values -- "A=", "02=", and "CHECK=". These names are used simply to illustrate that the values are not important as long as they are unique. You could just as easily use "A=", "B=", "C=" or "1=", "2=", "3=".

Also, note that after loading the INI file into memory the actual order of the list will be '02', then 'A' and finally 'CHECK' because INI files are sorted during loading. In this example, each function is independent of the others and the sorted order does not affect the program.

---

In this example, all three functions are in the same DLL, namely TMROS2, but this is not a requirement.

This example also demonstrates functions registered at STATE levels 0, 1 and 2 and URGENCY levels 0, 1, and 2. This is not a requirement. STATE levels and URGENCY levels can be mixed and matched to meet the service functions' requirements.

If necessary, you can register multiple service functions at the same STATE and URGENCY level -- even with the same time interval. Unless the DATA area is used by the service function to distinguish what action to take, it is not usually wise to register the same DLLName->FuncName more than once.

## Multiple Platforms

Remember [WINDOWSUBS] or [WINDOW32SUBS] entries may have to be added if the INI file is used by workstations operating on more than one platform. For instance, suppose a function is located in MINEOS2.DLL for OS/2 and MINEWIN.DLL for Windows. The correct INI entries might look like this:

```
[TIMERFUNCS]
01=;0;0;60;MINEOS2->MyFunction;
[WINDOWSUBS]
MINEOS2 = MINEWIN.DLL
```

Before attempting to load MINEOS2, all FSI programs will substitute the appropriate name from the [WINDOWSUBS] group. Establishing an INI file in this manner will make it useable by each supported environment. The same method is used for WIN32 programs using [WINDOW32SUBS].

Adding entries under [WINDOWSUBS] or [WINDOW32SUBS] is not usually necessary if the DLL that is being called is a FSI "base" DLL. The appropriate names for these DLLs for each platform are pre-registered for you.

## Timed Service Function Prototype

All service functions called by TMRLIB must conform to the FAPHANDLER function prototype. For more information on the FAPHANDLER prototype, see the [FAPHANDLER](#) section. This prototype takes the following form:

```
typedef DWORD (_VMMAPIPTR FAPHANDLER)(VMMHANDLE memH,  
                                       DWORD msgno,  
                                       FAPPARM p1,  
                                       FAPPARM p2);
```

Please note that this prototype is being used for convenience. The internal structures maintained for service functions are not true FAOBJECTs and do not receive “broadcast” messages as other FAOBJECTs do. This prototype contains the necessary parameters for service functions and we did not deem it necessary to make a new prototype name.

Your function definition should look something like the following (taken from TMRLIB).

```
DWORD _VMMAPI TMRTimerTest1(VMMHANDLE tmrH,  
                             DWORD msg,  
                             FAPPARM p1,  
                             FAPPARM p2)
```

The parameters passed to the service function have the following meanings:

Parameter	Description
VMMHANDLE tmrH	This variable will be the service function's VMMHANDLE in the timer list. This handle is not a descendant of FAOBJECT. The handle references the FSITIMERREC structure that defines the service function being called. Normally you will not need to address this structure unless the function requires implementation specific “data”, which is an element in the structure. To reference this structure it will be necessary to include TMRLIB.H.
DWORD msg	The requested operation message number. The following FAP messages are passed in the message parameter for the following operations.  FAP_MSGINIT This message is sent to service functions to perform initialization (if required).  FAP_MSGTERMINATE This message is sent to indicate that service functions should release all memory and resources that may be in use.  FAP_MSGRUN This message is sent when it is time for the service function to executes its task.  The definition of and implementation rules for these values is covered later in this section under the topic ???.
FAPPARM p1	This variable contains the handle of the main (client) window. Use (HWND)p1.dw to retrieve this value. Note, the HWND cast is necessary because a window handle is a 32-bit value under Windows 32-bit, but only a 16-bit value under Windows. Using the cast will eliminate compiler warnings.
FAPPARM p2	This variable contains a pointer to program specific data. The pointer can be retrieved by referencing this parameter as p2.ptr.  In Documaker Workstation, this pointer will be the AFEDATA structure. This value can be retrieved in the following manner.  PAFEDATA pdata = (PAFEDATA)p2.ptr;  The AFEDATA structure is defined in AFELIB.H. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care in manipulating the data within this structure.

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result. A non-successful return to the FAP\_MSGINIT message will result in the service function being disabled.

## Messages

As shown in the prototype section, there are three distinct messages passed to time service functions.

---

While the service function is being called, all other actions in the program are suspended. The function must return to allow processing to continue. Failure to return will cause the program to appear “dead”.

---

### *FAP\_MSGINIT*

This message is sent to all registered service functions at the time TMRLIB is initialized -- regardless of the current STATE or URGENCY variables.

If a value other than SUCCESS (0) is returned, the function is removed from the timer list and will not be called again.

Receiving this message is an indication that any initialization steps required to set up the function for timed callback may be performed. Initialization steps might include allocating memory, loading INI settings, variable initializations, and so on.

If SUCCESS is returned, the service function will be called each time the SECONDS values have elapsed and the STATE and URGENCY flags are satisfied.

### *FAP\_MSGTERMINATE*

This message is sent to all registered service functions (that returned SUCCESS during initialization) when TMRLIB is terminated. This may be due to program exit or due to some action (user or program) requesting that timers be terminated. Each function is called regardless of the STATE or URGENCY settings.

The value returned by the service function in response to this message is ignored.

Receiving this message is an indication that any termination steps required to end the service function's operation be performed. Termination steps might include closing files, freeing used memory or resources, closing windows, and so on.

After termination, it is possible for the service function to be called again with FAP\_MSGINIT to re-initialize the service function.

### *FAP\_MSGRUN*

Upon receipt of this message, the service function can perform the action for which it was designed. The value returned from the service function is ignored.

Receiving this message is an indication that the STATE and URGENCY values associated with the service function have been satisfied and the appropriate time (SECONDS ) has elapsed.

---

: Those functions that use URGENCY level 1 may have been delayed. This can be detected (if necessary) by checking the “activate” structure member associated with the service function.

---

## Considerations

Time is relative.

There is no assurance -- even when using a true OS timer -- that your function will be called on a precisely timed basis. This means that you should not expect to estimate how much time has elapsed simply based upon the time interval registered with the service function. If you need time information, you should use a clock function that returns the actual time-of-day.

The primary reason you cannot depend upon a precise time is that OS timers rely upon OS messages. If a program function spends excessive time doing a task without checking the program's message loop, the timer message will be delayed.

---

Keep this last point in mind. If your service functions take a long time to operate, it may interfere with the normal operation of the program. By returning quickly, you help to ensure that other functions, waiting on an "event" or message, will have their turn to act.

---

TMRLIB creates an actual OS timer that is activated approximately once each second. When the timer message is received, the service function list is checked to determine whether each function needs to be called using the current desktop state and the registered time interval (SECONDS) as filters. In addition, any service function that has been delayed -- due to an URGENCY state -- will be evaluated.

This STATE and URGENCY variables are intended to give writers of service functions some assurance that the manipulation of the form set (or potential form set) is safe and that the program is in a state that will allow the function to perform safely.

URGENCY level 2, however, remains active at all times. Service functions registered with this level may be called whether the user is actively engaged with menu options or windows. Be careful when writing a level 2 service function.

In addition to the possible delay caused by supporting STATE and URGENCY levels, service functions are called in a sequential fashion. If it is determined that more than one function needs to be called during a timer message, each service function must complete (and return) before the next can be called.

---

The number of service functions registered and the length of time it takes for each to complete its task can adversely affect program performance.

---

## Timing Example

```
[TIMERFUNCS]
01=:0;0;30;MINEOS2->TMRTimerTest1;
02=:1;0;60;MINEOS2->TMRTimerTest2;
```

The following analysis uses the example registrations shown above. These examples use URGENCY level 0 which means that the service functions can be skipped (entirely) if the open window and menu test fails.

If the user has not opened the desktop and is not engaged in a menu option window, the function TMRTimerTest1 will be called *approximately* every 30 seconds. As long as the desktop remains closed, the second function will not be called.

Now, assume 25 seconds pass without opening a form set. The user then opens the Form Selection window, spends 10 seconds mulling it over and finally cancels the window. Because the window was opened when the 30-second time interval occurred, the service function was not called. It will be another 25 seconds before the next evaluation is made to determine whether to run the function.

While the desktop is opened, the first service function will not be called. However, the countdown for the second service function will be activated. Approximately each 60 seconds (if a menu or window is not active), the second function will be called.

If after 59 seconds the user opens the Print window -- fiddles around for two seconds and then cancels -- it will take another 59 seconds before the second service function is checked again. On the other hand, if only 50 seconds have elapsed, the user opens the Print window for 9 seconds and then cancels. The 60-second timer will be able to execute the function `TMRTimerTest2`.

# Function and Hook Reference

---

# AddComment

Use this function to add a comment to the print stream. Products like Docucorp's Docusave and IBM's OnDemand use comments in the print stream as an archive key. The AddComment DAL function should only be called from a script loaded via the DocuSaveScript option specified in the AFP or Metacode printer control group. Calling AddComment from GenData will result in an INTERNAL ERROR being returned from DAL.

## Syntax:

AddComment (Comment, Convert)

Parameter	Description	Required?
Comment	A string to be written as a comment in the print stream	Yes
Convert	0 means convert string to EBCDIC (default) 1 means convert string to ASCII 2 means do not convert string For Docusave, you will always want EBCDIC comments.	No

## Example:

Here are some examples:

```
AddComment('This is an example')  
AddComment('@('INSURED NAME',,, GROUPNAME()))
```



---

# AFE Append Record Hook

## AFERetriB4AppendgToLstHook

This hook will be called immediately before a new record is added to the static list, after the OK button on the Retrieve window is pressed. The hook is expected to return a SUCCESS (0) or a FAILURE (a non-zero number). A FAILURE will cause the record not to be added to the list and a SUCCESS will cause the record to be added to the list. The type definition and prototype for this hook are listed below.

### Type Definition

```
typedef int (_VMMAPIPTR AFERETRIOKBUTTONHOOK)(GUIHWND hwnd, PAFEDATA pdata);
```

### Syntax

```
extern int _VMMAPI AnyFunctionName(GUIHWND hwnd, PAFEDATA pdata);
```

Parameter	Description
hwnd	The handle to the window which calls the function AFERetDisplayList.
pdata	A pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. The values entered into Key1, Key2 and KeyID on the Retrieve window are stored in the WIP record structure that is a member of the pdata structure (pdata->WIP). The search on the database is base on those values.

### INI Definition

```
[ AfeProcedures ]
AFERetriB4AppendgToLstHook = DLL->FunctionName
```

The (*DLL*) is the DLL where the function hook is. The (*FunctionName*) is the function name of the hook. For more information on this subject, read the section above on: “How does Documaker Workstation locate external procedures”.

### Important Information

To avoid memory errors, the memory location allocated for the WIP record should not be freed. That process will be taken care off after returning from the hook.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# AFE Archive List Hook

## AFERetDisplLstHook

This hook will replace the AFERetDisplayList() function. The replaced function is primarily responsible for retrieving records from the archive database and adds them to a static list. The prototype and type definition of this function is listed below.

### Type Definition

```
typedef int (_VMMAPIPTR AFERETRIDISPLHOOK)(HWND      hwnd,
PAFEDATA  pdata,
VMMHANDLE AppListH,
BOOL      bFirst);
```

### Syntax

```
extern int _VMMAPI AnyFunctionName(  HWND      hwnd,
PAFEDATA  pdata,
VMMHANDLE AppListH,
BOOL      bFirst);
```

Parameter	Description
Hwnd	The handle to the window which calls the function AFERetDisplayList.
Pdata	A pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. The values entered into Key1, Key2 and KeyID on the Retrieve window are stored in the WIP record structure that is a member of the pdata structure (pdata->WIP). The search on the database is base on those values.
AppListH	A static pointer. This pointer points to the list of records display at the Retrieve window. To avoid a memory leak, AppListH should be destroyed every time a new list is built.
Bfirs	Serves as a guild to the previous parameter. Since this function can be called more than one time to build a list, ( <i>bFirst</i> ), a Boolean, tells the function when it is been called for the first time during the process of building the current list.

### INI Definition

```
[ AfeProcedures ]
AfeRetDisplLstHook = DLL->FunctionName
```

The (*DLL*) is the DLL where the function hook is. The (*FunctionName*) is the function name of the hook. For more information on this subject, read the section above on: “How does Documaker Workstation locate external procedures”.

### Important Information

To avoid memory errors, the memory location allocated for the WIP record should not be freed. That process will take place after returning from the hook.

### *Remarks*

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# AFE Archive Record Selected Hook

## AFERetriOkButtonHook

This hook will be called immediately after the OK button from the Retrieve window is pressed. The hook is expected to return a SUCCESS (0) or a FAILURE (a non-zero number). A FAILURE will cause the process to return to the window, and a SUCCESS will cause the process to continue normally. The type definition and prototype for this hook are listed below.

### Type Definition

```
typedef int (_VMMAPIPTR AFERETRIOKBUTTONHOOK)(GUIHWND hwnd, PAFEDATA pdata);
```

### Syntax

```
extern int _VMMAPI AnyFunctionName(GUIHWND hwnd, PAFEDATA pdata);
```

Parameter	Description
hwnd	The handle to the window which calls the function AFERetDisplayList.
pdata	A pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. The values entered into Key1, Key2 and KeyID on the Retrieve window are stored in the WIP record structure that is a member of the pdata structure (pdata->WIP). The search on the database is base on those values.

### INI Definition

```
[ AfeProcedures ]  
AFERetriOkButtonHook = DLL->FunctionName
```

The (*DLL*) is the DLL where the function hook is. The (*FunctionName*) is the function name of the hook. For more information on this subject, read the section above on: "How does Documaker Workstation locate external procedures".

### Important Information

To avoid memory errors, the memory location allocated for the WIP record should not be freed. That process will take place after returning from the hook.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# AFE Check Form Set Data Hook

## CheckUserEntry

This hook is called before a new WIP entry can be created. No default value is assumed for this option.

### INI Definition

```
[ AFEProcedures ]  
CheckUserEntry  = DLL->FuncName
```

### Syntax

```
int _VMMAPI func( HWND hwnd, char *Key1, char *Key2, char *KeyID,  
char *Desc, VMMHANDLE formsetH, PAFEDATA pdata);
```

Parameter	Description
hwnd	The handle to the Form Selection window or zero.
Key1	A pointer to a NULL terminated string that represents the Key1 value.
Key2	A pointer to a NULL terminated string that represents the Key2 value.
KeyID	A pointer to a NULL terminated string that represents the KeyID value.
Desc	A pointer to a NULL terminated string that represents the WIP description.
FormsetH	A handle to the current form set.
Pdata	A pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result. A non-successful return prevents the WIP from being created.

---

# AFE Complete Form Set Hook

## Complete

Support for this hook began in July, 1996. There is no default for this option.

When the user takes the “Complete” action, this is an indication that the form set should be examined for valid entries. Upon a successful field check, the Complete Form Set window appears, this lets the user print, export, and archive the form set.

This hook is called at three points in this process. This lets the system add the customized features to Complete.

### INI Definition

```
[ AFEProcedures ]  
Complete = DLL->FuncName
```

### Syntax

This function must conform to the FAPUSER prototype. For more information on the FAPUSER prototype, see the [FAPUSER](#) section. You can use the following parameters:

Parameter	Description
DwMessage	<p>A message requesting a particular operation. The following FAP messages are passed in the dwMessage parameter for the following operations, and should be handled accordingly in a custom procedure. Please note that although FAP message numbers are being used, there is no FAPOBJECT that initiates or receives the action.</p> <p><b>FAP_MSGINIT</b> This message is sent before any internal verification of the form set data and before the Complete window appears.</p> <p><b>FAP_MSGRUN</b> This message is sent after the user has pressed OK on the Complete window and before the default functionality is executed.</p> <p><b>FAP_MSGTERMINATE</b> This message is sent after the Complete window has been removed and the user chose OK.</p>
DwFAPHab	The program's anchor block or instance handle. The distinction depends upon whether the program is running on an OS/2 or Windows platform. Within the Docucorp programming environment, both definitions serve the same purpose.
DwFAPHwnd	A handle to the currently open window. Only on the message, FAP_MSGRUN, will this handle represent the Complete window.
DwObjectIdentifier	Not used.
DwObjectType	Not used.
DwInputFlag1	Not used.
DwInputFlag2	Not used.
DwInputFlag3	Not used.
LpszUserID	A pointer to a NULL terminated string that contains the user ID of the current

	operator.
LpszTranCode	A pointer to a NULL terminated string that contains the Transaction Code associated with the WIP record.
lpszKey1	A pointer to a NULL terminated string that contains the Key1 field value associated with the WIP record.
lpszKey2	A pointer to a NULL terminated string that contains the Key2 field value associated with the WIP record. Note that in a multi-select situation (PPS) only the first Key2 value is provided.
LpszInputBuffer	A pointer to a NULL terminated string that contains the current KeyID field value associated with the WIP record.
LpszKeyID	A pointer to a text buffer that should receive the output KeyID from this function. If this value is NULL, no output string is expected. On input, a non-NULL value will represent the last KeyID returned from your hook procedure. A difference between the input KeyID and the one represented in this string means that the user changed the original KeyID.
DwOutputBufferMaxSize	Not used.
PAFEData	A pointer the current AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
LpdwOutputFlag2	Not used.
LpdwOutputFlag3	Not used.

### *Remarks*

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether each operation succeeded.

In response to the message, FAP\_MSGINIT, a non-successful return will cause the complete function to return unsuccessful. The custom function should tell the user what error occurred and what to do next.

In response to the message, FAP\_MSGRUN, a non-successful return will not execute the default functionality associated with the OK action and the window will remain active. It is possible for a custom function to remove the window manually if so desired.

The response to the message FAP\_MSGTERMINATE does not affect the completion process in any way, as it is sent after the default functionality has executed.

---

# AFE Entry Form Set Hook

## EntryFormset

Support for this hook began in April, 1996. No default INI value is assumed for this option.

A form set appears on the user's screen and a series of functions are performed. For instance, the form set is filtered by removing any forms that are not selected and "Required" forms are checked for inclusion. This hook was created to allow customization within this process.

Three calls (with separate messages) are made to the hook each time a form set is loaded.

### INI Definition

```
[ AFEProcedures ]
EntryFormset = DLL->FuncName
```

### Syntax

This function must conform to the FAPUSER prototype. For more information on the FAPUSER prototype, see the [FAPUSER](#) section. You can use the following parameters:

Parameter	Description
DwMessage	A message requesting a particular operation. The following FAP messages are passed in the dwMessage parameter for the following operations, and should be handled accordingly in a custom procedure.  FAP_MSGCREATE This message is sent before any internal verification of the form set is performed. In this state, the form set may contain more forms than the user selected and will not have filtered out forms that should not be displayed.  FAP_MSGOPEN This message is sent after the form set has been prepared and before loading the images. At this point, only those forms that remain will be loaded or displayed.  FAP_MSGRUN This message is sent after the form set is loaded and before the form set begins display for Entry.
DwFAPHab	Not used.
DwFAPHwnd	Not used.
DwFormsetH	Handle of the current form set.
DwObjectType	Is FAP_OBJFORMSET.
DwInputFlag1	Not used.
DwInputFlag2	Not used.
DwInputFlag3	Not used.
LpszUserID	Not used.
LpszTranCode	Not used.
lpszKey1	Not used.



lpszKey2	Not used.
PAFEData	A pointer the current AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
LpszOutputBuffer	Not used.
DwOutputBufferMaxSize	Not used.
LpdwOutputFlag1	Not used.
LpdwOutputFlag2	Not used.
LpdwOutputFlag3	Not used.

### *Remarks*

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether each operation succeeded. For each message, a non-successful return prevents the form set from being displayed.

---

# AFE Form Selection Buttons Hook

## BUTTONx

Supported after July 1996, three buttons can be activated on the Form Selection window when associated with custom functions. No default INI value is assumed for these buttons. Any button that does not have a registered function is hidden from view when the Form Selection window appears.

These buttons may be used for any activity related to form selection. Two standard procedures are provided in the base product that may be used. TRNAutoNextKey() will remove any existing KeyID and request a new KeyID from the AutoKeyID function. AFEPersonalEdit() will activate the “Personal Form Selection” window and allow the user to specify a subset of forms to display as a personal Key1-Key2 combination.

### INI Definition

```
[ AFEProcedures ]
BUTTON1 = TRNOS2->TRNAutoNextKey
BUTTON2 = AFEOS2->AFEPersonalEdit
BUTTON3 = TSTOS2->UserTest
```

Note that you do not have to define any or all of these buttons. You may activate BUTTON3 without activating buttons 1 or 2, and so on.

### Syntax

This function must conform to the FAPUSER prototype. For more information on the FAPUSER prototype, see the [FAPUSER](#) section. You can use the following parameters:

Parameter	Description
DwMessage	A message requesting a particular operation. The following FAP messages are passed in the dwMessage parameter for the following operations, and should be handled accordingly in a custom procedure. Please note that although FAP message numbers are being used, there is no FAPOBJECT that initiates or receives the action.  FAP_MSGINIT Perform initialization, if necessary. Assign custom button text (default text is button name).  FAP_MSGRUN Button was activated by user. Perform task.  FAP_MSGTERMINATE Perform termination and cleanup, if necessary.
DwFAPHab	The program's anchor block or instance handle. The distinction depends upon whether the program is running on an OS/2 or Windows platform. Within the Docucorp programming environment, both definitions serve the same purpose.
DwFAPHwnd	A handle to the currently open window. In all cases, this should be the handle of the Form Selection window.
DwChildID	The ID of a child control on the window that initiated the call or zero if no child ID is available.
DwObjectType	Not used.
DwInputFlag1	Not used.

DwInputFlag2	Not used.
DwInputFlag3	Not used.
LpszUserID	A pointer to a NULL terminated string that contains the User ID of the current operator.
LpszTranCode	A pointer to a NULL terminated string that contains the Transaction Code associated with the WIP record.
lpszKey1	A pointer to a NULL terminated string that contains the Key1 field value associated with the WIP record.
lpszKey2	A pointer to a NULL terminated string that contains the Key2 field value associated with the WIP record. Note that in a multi-select situation (PPS) only the first Key2 value is provided.
LpszKeyID	A pointer to a NULL terminated string that contains the current KeyID field value associated with the WIP record.
LpszOrigKeyID	A pointer to a NULL terminated string that contains the original KeyID. A difference between the input KeyID and the one represented in this string means that the user changed the original KeyID. No output string is expected from the custom function.
DwOutputBufferMaxSize	Represents the maximum size of the output buffer for the previous parameter. however, any value copied to the <i>char FAR *lpszOutputBuffer</i> is ignored.
PAFEData	A pointer the current AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
LpdwOutputFlag2	Not used.
LpdwOutputFlag3	Not used.

### Remarks

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded. In response to the message FAP\_MSGINIT, a non-successful return will cause the button to be hidden from view and deactivated.

The parameters provided to this hook function includes all the current WIP information required to understand how the user constructed (or is constructing) the form set. A custom function may or may not use the information provided.

---

# AFE Initialization Hook

## Init

This hook is called any time the INI settings are loaded. Usually this occurs at the program start and when master resources change. There is no default for this option.

### INI Definition

```
[ AFEProcedures ]
Init = DLL->FuncName
```

### Syntax

```
int _VMMAPI func( HAB hab, PAFEDATA pdata);
```

Parameter	Description
hab	The program's anchor block or instance handle. The distinction depends upon whether the program is running on an OS/2 or Windows platform. Within the Docucorp programming environment, both definitions serve the same purpose.
Pdata	A pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.

### Remarks

The function should return SUCCESS (0) or FAIL (not zero) to indicate the result of the operation.

This hook is called after loading INI values so variables in the AFEDATA structure will reflect current settings.

Note that this function can be called more than once in a session. A custom function that only wants to be called once should keep track of this fact or remove the INI setting after being called the first time.

---

# AFE Parse Command Line Hook

## Parse

This hook is called to allow the command line parameters to be parsed and used in custom code rather than base code. The actual values can be found in AFEData->argv. With this hook, system values can be changed based on parameter values.

### INI Definition

```
[ AFEProcedures ]  
Parse = DLL->FuncName
```

### Syntax

```
DWORD _VMMAPI CUSParse(AFEDATA *AFEData);
```

Parameter	Description
AFEData	Contains the current AFEDATA structure. This should contain all material that is needed for information as well as values for changing.

### Remarks

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded.

A call to this function is an indication that Operations are reliant on command line parameters being translated into values or actions.

The effects of the parsing are determined by the return value. Returning SUCCESS will indicate that the program should continue. Returning FAIL will halt program execution. For example, if a password is expected on the command line, program execution can be halted when a password is not found by returning FAIL.

---

# AFE Post Edit Hook

## PostEdit

This hook is called before a form set is saved (unloaded), assigned to another user, completed, or deleted. No default value is assigned to this option.

Although this function would appear to be the counterpart of the *PREEDIT* hook, it can be called in many more circumstances. A custom function should check the AFEDATA structure to determine in what mode the form set is being treated -- AFEACTION\_UPDATE, AFEACTION\_DELETE, or AFEACTION\_COMPLETE.

### INI Definition

```
[ AFEProcedures ]
PostEdit = DLL->FuncName
```

### Syntax

Transaction functions must conform to the FAPHANDLER prototype. For more information on the FAPHANDLER prototype, see the [FAPHANDLER](#) section. You can use the following parameters:

Parameter	Description
<i>ObjectH</i>	The handle of the current form set.
<i>msgno</i>	A requested operation message number. Currently, only the FAP_MSGPOSTEDIT message is sent.
p1	Is a union. p1.dwr will be the AFEDATA structure value action.
p2	Is a union. p2.ptr is a pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result. However, only before the form set being saved will a non-successful return prevent the action from being taken.

---

# AFE Pre Edit Hook

## PreEdit

This hook is called after a form set has been loaded successfully and before Entry can begin. No default value is assumed for this option.

The AFEDATA structure should be queried to determine in what mode the form set has been opened -- AFEACTION\_CREATE, or AFEACTION\_UPDATE.

### INI Definition

```
[ AFEProcedures ]  
PreEdit = DLL->FuncName
```

### Syntax

Transaction functions must conform to the FAPHANDLER prototype. For more information on the FAPHANDLER prototype, see the [FAPHANDLER](#) section. You can use the following parameters:

Parameter	Description
<i>ObjectH</i>	The handle of the current form set.
<i>msgno</i>	A requested operation message number. Currently, only the FAP_MSGPREEDIT message is sent.
<i>p1</i>	Is a union. p1.ptr will be a pointer the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
<i>p2</i>	Is a union. p2.ptr is a pointer to the TRANSREC structure associated with the form set.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result. A non-successful return indicates that the form set should not be edited.

---

# AFE Termination Hook

## Term

This hook is called when the Entry module is exiting. A custom function should release any resources or memory that were allocated during program execution when this hook is called. There is no default for this hook.

## INI Definition

```
[ AFEProcedures ]  
Term = DLL->FuncName
```

## Syntax

```
int _VMMAPI func( HAB hab, PAFEDATA pdata);
```

Parameter	Description
hab	Is the program's anchor block or instance handle. The distinction depends upon whether the program is running on an OS/2 or Windows platform. Within the Docucorp programming environment, both definitions serve the same purpose.
pdata	A pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.

## Remarks

The return value from this call is ignored. This hook is called before releasing the AFEDATA structure members.



---

# AFE Window Procedure Hook

## WindowProc

This hook is designed to let a custom function intercept messages that arrive at the main application's window procedure. There is no default for this hook.

A custom procedure has first use of incoming messages and can indicate whether the default message handling should continue.

### INI Definition

```
[ AFEProcedures ]  
WindowProc = DLL->FuncName
```

### Syntax

This function should conform to the WNDPROC typedef prototype. This definition is as follows:

```
DWORD _VMMAPI func(HWND hwnd, MSG msg, WPARAM mp1, LPARAM mp2,  
    LRESULT *result, void *data);
```

Parameter	Description
hwnd	The window handle (usually) passed to the WNDPROC.
msg	The window message indicator passed to the WNDPROC.
mp1	The first window parameter passed to the WNDPROC.
mp2	The second window parameter passed to the WNDPROC.
result	A pointer to a long value that should contain the value to return if further processing is not to continue.
data	A pointer to the AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.

### Remarks

The function should return SUCCESS (0) if the custom function handled the message and no further processing is to continue. A non-zero return means that default processing should continue on the message.

When SUCCESS is returned the *result* assigned by the custom function will be returned to the caller.

---

# AFEArchive2WipKeys

## Archive2WIP

In most instances, the archive file structures will be based upon the WIP file structures in use by the program. Since the Entry module supports the ability to customize archive files, two hooks are provided to translate the Key1, Key2, and KeyID components typically used by the WIP and Archive/Retrieval system.

ARCHIVE2WIP is called to translate the archive Key1, Key2, and KeyID fields into the corresponding WIP fields. Another hook, WIP2ARCHIVE, exists to translate in the other direction.

There is no default for this hook, therefore you must register this function or archive retrieval cannot be accomplished.

If a custom procedure is not provided for AFE, the procedure AFEArchive2WipKeys() must be defined to perform this task.

### INI Definition

```
[ AFEProcedures ]
Archive2WIP = AFEOS2->AFEArchive2WipKeys
```

### Syntax

```
int _VMMAPI AFEArchive2WipKeys(char * inKey1,
                                char * inKey2,
                                char * inKeyId,
                                char * outKey1,
                                char * outKey2,
                                char * outKeyId);
```

The parameters of this function should be self-explanatory. The input versions of Key1, Key2, and KeyID are represented in the first three parameters. The translated versions of these fields should be copied to the corresponding out fields.

Parameter	Description
inKey1	Input Key1 from Archive.
inKey2	Input Key2 from Archive.
InKeyId	Input KeyID from Archive.
outKey1	Output Key1 for WIP. The inKey1 value should be translated into this field.
outKey2	Output Key2 for WIP. The inKey2 value should be translated into this field.
OutKeyId	Output KeyID for WIP. The InKeyId value should be translated into this field.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

As the name should imply, this function should convert archive key field information into WIP key field information.

---

# AFESecurityFunc

## Security

This hook is called to verify a user has access rights to enter the program and when the user logs out of the program. The default function, AFESecurityFunc(), is used if no security value is defined.

### INI Definition

```
[ AFEProcedures ]  
Security = AFEOS2->AFESecurityFunc
```

### Syntax

This function must conform to the FAPUSER prototype. For more information on the FAPUSER prototype, see the [FAPUSER](#) section. You can use the following parameters:

Parameter	Description
DwMessage	A message requesting a particular operation. The following FAP messages are passed in the dwMessage parameter for the following operations, and should be handled accordingly in a custom procedure.  FAP_MSGINIT This message indicates that a user wants to log into the program.  FAP_MSGTERMINATE This message indicates that a user wants to log out of the program.
DwFAPHab	Is the program's anchor block or instance handle. The distinction depends upon whether the program is running on an OS/2 or Windows platform. Within the Docucorp programming environment, both definitions serve the same purpose.
DwFAPHwnd	A handle to the currently open window.
DwObjectIdentifier	Not used.
DwObjectType	Not used.
DwInputFlag1	Not used.
DwInputFlag2	Not used.
DwInputFlag3	Not used.
LpszUserID	A pointer to a NULL terminated string that represents any known user ID. This value may have been obtained as a parameter from the program's command line or the result of querying the value for the UserID option in the SignOn control group.
LpszFormatType	Not used.
LpszFormat	Not used.
LpszFileName	A pointer to a NULL terminated string that represents a file name obtained by combining the values for Path and File options in the UserInfo control group.
PAFEData	A pointer the current AFEDATA structure used by AFE. This structure contains most if not all of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.

LpszOutputBuffer	A pointer to a USERREC structure that should be filled by the custom function.
DwOutputBufferMaxSize	Is the size of the USERREC structure.
LpdwOutputFlag1	Not used.
LpdwOutputFlag2	Not used.
LpdwOutputFlag3	Not used.

### *Remarks*

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the requested operation succeeded.

On FAP\_MSGINIT (logon), a non-successful return will prevent the user from obtaining access to the program.

After a successful logon attempt, the “userlistH” member of the AFEDATA structure is examined. This member is a VMM list containing all user records (USERREC structures) that report to this current user (including this user).

If it has not been assigned data by the custom function, the user file is examined for the data. If no user file is found, only the current user record is added to the reports to list.

---

# AFEWip2Archive

## Wip2Archive

In most instances, the archive file structures will be based upon the WIP file structures in use by the program. Since the Entry module supports the ability to customize archive files, two hooks are provided to translate the Key1, Key2, and KeyID components typically used by the WIP and Archive/Retrieval systems.

WIP2ARCHIVE is called to translate the Key1, Key2, and KeyID fields into the corresponding archive key fields. Another hook, ARCHIVE2WIP, exists to translate in the other direction.

There is no default for this hook, therefore you must register this function or archive retrieval cannot be accomplished.

If a custom procedure is not provided for AFE, the procedure AFEWip2ArchiveKeys() must be defined to perform this task.

### INI Definition

```
[ AFEProcedures ]  
Wip2Archive = AFEOS2->AFEWip2ArchiveKeys
```

### Syntax

```
int _VMMAPI AFEWip2ArchiveKeys(char * inKey1,  
                                char * inKey2,  
                                char * inKeyId,  
                                char * outKey1,  
                                char * outKey2,  
                                char * outKeyId);
```

The parameters of this function should be self-explanatory. The input versions of Key1, Key2, and KeyID are represented in the first three parameters. The translated versions of these fields should be copied to the corresponding out fields.

Parameter	Description
inKey1	Input Key1 from WIP.
inKey2	Input Key2 from WIP.
InKeyId	Input KeyID from WIP.
outKey1	Output Key1 for Archive. The inKey1 value should be translated into this field.
outKey2	Output Key2 for Archive. The inKey2 value should be translated into this field.
OutKeyId	Output KeyID for Archive. The InKeyId value should be translated into this field.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result. As the name should imply, this function should convert WIP key field information into Archive key field information

---

# AFEWip2ArchiveRecord

## Archive

Since the Entry module supports custom Archive Index files, a hook is provided to create the archive index record from a WIP record. No default INI definition is assumed, therefore this function must be registered or archiving cannot be accomplished.

If a custom procedure is not provided for AFE, the procedure AFEWip2ArchiveRecord() must be defined to perform this task. This function uses another INI group to map the fields from the WIP record to the fields of an archive record.

### INI Definition

```
[ AFEProcedures ]
Archive = AFEOS2->AFEWip2ArchiveRecord
```

### Syntax

```
int _VMMAPI AFEWip2ArchiveRecord(PAFEDATA pdata,
                                  VMMHANDLE wipdfdH,
                                  void FAR *wiprec,
                                  VMMHANDLE arcdfdH,
                                  void FAR *arcrc);
```

Parameter	Description
Pdata	A pointer the AFEDATA structure defined in AFELIB.H. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
wipdfdH	A handle to the data file definition (DFD) file that identifies all the fields and their types used in a WIP record.
wiprec	A record buffer that contains the current WIP information.
arcdfdH	A handle to the DFD file that identifies all the fields and their types used in an archive index record.
arcrc	An index record buffer that is the destination of the converted information.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

This function is called any time a WIP record is to be converted to an archive index record. This function should limit its activities to that described since not all calls will result in an actual archive file or index record being written or retrieved.

---

# AppIdxRec

Use this function to get an archive record based on APPIDX.DFD and Trigger2Archive INI settings.

*Syntax:*

```
AppIdxRec ()
```

*Example:*

```
Comment = AppIdxRec()  
AddComment( Comment )
```

---

# CUSGetArcIdxName

## IndexName

This function is only supported in version 9.0 or greater of the Development System/Docucreate.

The Entry module supports the ability to customize archive files. In addition to the other hooks that support translation of information between the WIP and archive systems, this hook is used to return the index file name used for archiving (or retrieving) a specific WIP record.

There is no default for this function. A standard function, CUSGetArcIdxName(), is provided as a basis for defining custom index names.

### INI Definition

```
[ AFEProcedures ]
INDEXNAME = CUSOS2->CUSGetArcIdxName
```

### Syntax

```
DWORD _VMMAPI CUSGetArcIdxName(VMMHANDLE dfdH,
                                void * record,
                                char * outIdxName,
                                size_t stFileNameSz);
```

Parameter	Description
DfdH	A DB type handle to a DFD file that defines the data record.
Record	A data buffer that contains current record information.
OutIdxName	An output buffer that should receive the file name of an index file to use.
StFileNameSz	The maximum size of the output field, outIdxName.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

The custom procedure attached to this hook may use the WIP record passed or any other information available to determine what index file name to return.



---

# DSDefAppendBuffer

## Append

This hook is called to append a buffer of data to the current file that was opened by a call to [Open](#) or [Create](#).

Document set files includes the NA, POL and PKG files. Default functionality is provided for reading and writing these files as ordinarily DOS files if another INI option is not provided. These functions can be overridden to achieve an alternate method of reading and writing this information.

## INI Definition

Two separate definitions are used to distinguish between reading document files in archive mode and non-archive mode. This was designed in a manner to assist those that might require their archived information to be retrieved from another location or in different manner than WIP files.

```
[ AFEDSProcedures ]  
APPEND = DSOS2->DSDefAppendBuffer  
[ AFEDSArchiveProcedures ]  
APPEND = DSOS2->DSDefAppendBuffer
```

## Syntax

```
DWORD _VMMAPI DSDefAppendBuffer(char *buffer, BOOL eof);
```

Parameter	Description
Buffer	Represents the NULL terminated text data that should be written.
eof	Will be TRUE to indicate that buffered data should be “flushed” to make sure it is written before CLOSE is called.

## Remarks

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded.

This function should write the data to the current file at the current offset. Since DSLIB uses text information rather than binary, you must translate the information if the specified file is not in that format.

You must determine what task to perform to write to the requested file. You must also maintain a link to the current open file since no handle or pointer is provided as a parameter.

---

# DSDefCloseBuffer

## Close

This hook is called to close the current file that was opened by a call to Open or Create.

Generally, at the end of file use or if an error occurs after successfully opening the specified file, DSLIB will call the registered CLOSE function.

Document set files includes the NA, POL and PKG files. Default functionality is provided for reading and writing these files as ordinarily DOS files if another INI option is not provided. These functions can be overridden to achieve an alternate method of reading and writing this information.

## *INI Definition*

Two separate definitions are used to distinguish between closing document files in archive mode and non-archive mode. This was designed in a manner to assist those that might require their archived information to be retrieved from another location or in different manner than WIP files.

```
[ AFEDSProcedures ]  
CLOSE = DSOS2->DSDefCloseBuffer  
[ AFEDSArchiveProcedures ]  
CLOSE = DSOS2->DSDefCloseBuffer
```

## *Syntax*

```
DWORD _VMMAPI DSDefCloseBuffer(void);
```

## *Remarks*

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded.

A call to this function is an indication that any currently open file be closed. You must determine what task to perform to close the requested file. You must also maintain a link to the current open file since no handle or pointer is provided as a parameter.

---

# DSDefCreateBuffer

## Create

This hook is called to create a new or truncate an existing file for writing.

Document set files includes the NA, POL and PKG files. Default functionality is provided for reading and writing these files as ordinarily DOS files if another INI option is not provided. These functions can be overridden to achieve an alternate method of reading and writing this information.

## INI Definition

Two separate definitions are used to distinguish between creating document files in archive mode and non-archive mode. This was designed in a manner to assist those that might require their archived information to be retrieved from another location or in different manner than WIP files.

```
[ AFEDSProcedures ]  
CREATE = DSOS2->DSDefCreateBuffer  
[ AFEDSArchiveProcedures ]  
CREATE = DSOS2->DSDefCreateBuffer
```

## Syntax

```
DWORD _VMMAPI DSDefCreateBuffer(char *filename);
```

Parameter	Description
<i>Filename</i>	The requested name of a file to create with any required path and extension applied (if system defined).

## Remarks

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded.

A call to this function is an indication that the requested file should be opened for writing. Nothing will be read from this file. You must determine what task to perform to “create” the requested file. You must also determine what and how to maintain a link to the current open file since no handle or pointer is returned to the calling function.

Generally, at the end of file use or if an error occurs after successfully opening the specified file, DSLIB will call the registered CLOSE function.

---

# DSDefFirstBuffer

## First

This hook is called to return the first buffer of data from the current file that was opened by a call to [Open](#).

Document set files includes the NA, POL and PKG files. Default functionality is provided for reading and writing these files as ordinarily DOS files if another INI option is not provided. These functions can be overridden to achieve an alternate method of reading and writing this information.

## INI Definition

Two separate definitions are used to distinguish between reading document files in archive mode and non-archive mode. This was designed in a manner to assist those that might require their archived information to be retrieved from another location or in different manner than WIP files.

```
[ AFEDSProcedures ]  
FIRST = DSOS2->DSDefFirstBuffer  
[ AFEDSArchiveProcedures ]  
FIRST = DSOS2->DSDefFirstBuffer
```

## Syntax

```
DWORD _VMMAPI DSDefFirstBuffer(char *buffer, size_t buffersize);
```

Parameter	Description
Buffer	Represents where the data should be placed after reading.
Buffersize	Indicates the maximum size of buffer.

## Remarks

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded. A return value of FAIL is assumed to mean EOF, since the file must have been opened successfully before making this call. Returning FAIL also means that no data was placed in the buffer.

This hook function should seek to the beginning of the current file (if necessary) and read the first *buffersize* bytes. Since DSLIB expects to receive text information rather than binary, you must translate the information if the specified file is not in that format. The buffer should be NULL terminated at the end of the text data that was read.

You must determine what task to perform to seek and read the requested file. You must also maintain a link to the current open file since no handle or pointer is provided as a parameter.

---

# DSDefNextBuffer

## Next

This hook is called to return the next buffer of data from the current file that was opened by a call to [Open](#).

Document set files includes the NA, POL and PKG files. Default functionality is provided for reading and writing these files as ordinarily DOS files if another INI option is not provided. These functions can be overridden to achieve an alternate method of reading and writing this information.

## INI Definition

Two separate definitions are used to distinguish between reading document files in archive mode and non-archive mode. This was designed in a manner to assist those that might require their archived information to be retrieved from another location or in different manner than WIP files.

```
[ AFEDSProcedures ]  
NEXT = DSOS2->DSDefNextBuffer  
[ AFEDSArchiveProcedures ]  
NEXT = DSOS2->DSDefNextBuffer
```

## Syntax

```
DWORD _VMMAPI DSDefNextBuffer(char *buffer, size_t buffersize);
```

Parameter	Description
Buffer	Represents where the data should be placed after reading.
Buffersize	Indicates the maximum size of buffer.

## Remarks

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded. A return value of FAIL is assumed to mean EOF, since the file must have been opened successfully before making this call. Returning FAIL also means that no data was placed in the buffer.

This hook function should begin reading of the current file at the current offset and read the next *buffersize* bytes. Since DSLIB expects to receive text information rather than binary, you must translate the information if the specified file is not in that format. The buffer should be NULL terminated at the end of the text data that was read.

You must determine what task to perform to read the requested file. You must also maintain a link to the current open file since no handle or pointer is provided as a parameter.

---

# DSDefOpenBuffer

## Open

This hook is called to open an existing file for reading.

Document Set files includes the NA, POL and PKG files. Default functionality is provided for reading and writing these files as ordinarily DOS files if another INI option is not provided. These functions can be overridden to achieve an alternate method of reading and writing this information.

### *INI Definition*

Two separate definitions are used to distinguish between opening document files in archive mode and non-archive mode. This was designed in a manner to assist those that might require their archived information to be retrieved from another location or in different manner than WIP files.

```
[ AFEDSProcedures ]  
OPEN = DSOS2->DSDefOpenBuffer  
[ AFEDSArchiveProcedures ]  
OPEN = DSOS2->DSDefOpenBuffer
```

### *Syntax*

```
DWORD _VMMAPI DSDefOpenBuffer(char *filename);
```

Parameter	Description
Filename	The requested name of a file to open with any required path and extension applied (if system defined).

### *Remarks*

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded.

A call to this function is an indication that the requested file should be opened for reading. Nothing is written to this file. You must determine what task to perform to open the requested file. You must also determine what and how to maintain a link to the current open file since no handle or pointer is returned to the calling function.

Generally, at the end of file use or if an error occurs after successfully opening the specified file, DSLIB will call the registered CLOSE function.

---

# GVM

Use this function to get the contents of a GVM variable.

*Syntax:*

GVM (VarName, Instance)

Parameter	Description	Required?
VarName	String containing the name of the GVM variable	Yes
Instance	Instance number of the GVM variable (default = 1)	No

*Example:*

Here is an example:

```
IF (HaveGVM('Company'))
  AddComment( GVM('Company') )
END
```

---

# HaveGVM

Use this function to see if a GVM variable exists.

*Syntax:*

HaveGVM (VarName, Instance)

Parameter	Description	Required?
VarName	String containing the name of the GVM variable	Yes
Instance	Instance number of the GVM variable (default = 1)	No

*Example:*

Here is an example:

```
IF (HaveGVM('Company'))
  AddComment( GVM('Company') )
END
```



---

# LBYCARRetrieveFile

## RetrieveFile

### INI Definition

```
[ VCS ]  
RetrieveFile = LBYOS2->LBYPARRetrieveFile
```

### Syntax

```
LONG _VMMAPI LBYPARRetrieveFile(void * indexrec,  
                                char *libname,  
                                char *filename,  
                                char *sequence);
```

Parameter	Description
Indexrec	An index record; use this to get version #, user ID, and so on, if any of this information is needed.
libname	Library to store file in.
filename	File to save to or retrieve from.
sequence	A key identifying the file being saved/retrieved. In Save, this can be set to whatever key the lower level code returns. The key will be saved, and passed into later calls to Retrieve.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LBYCARRetrieveMemFile

## RetrieveMemFile

### INI Definition

```
[ VCS ]  
RetrieveMemFile = LBYOS2->LBYPARRetrieveMemFile
```

### Syntax

```
LONG _VMMAPI LBYPARRetrieveMemFile(void * indexrec,  
                                     char *libname,  
                                     char *filename,  
                                     char *sequence);
```

Parameter	Description
indexrec	An index record; use this to get version #, user ID, and so on, if any of this information is needed.
libname	Library to store file in.
filename	File to save to or retrieve from.
sequence	A key identifying the file being saved or retrieved. In Save, this can be set to whatever key the lower level code returns. The key will be saved, and passed into later calls to Retrieve.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LBYCARSaveFile

## SaveFile

### INI Definition

```
[ VCS ]  
SaveFile = LBYOS2->LBYCARSaveFile
```

### Syntax

```
LONG _VMMAPI LBYSaveFile(void *indexrec,  
                           char *libname,  
                           char *filename,  
                           char *key)
```

Parameter	Description
indexrec	An index record; use this to get the version number, user ID, and so on, if any of this information is needed.
libname	Library to store file in.
filename	File to save to or retrieve from.
key	A key identifying the file being saved or retrieved. In Save, this can be set to whatever key the lower level code returns. The key will be saved, and passed into later calls to Retrieve.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LMGLBYCheckin

## CheckIn

Called when user wants to put back changes.

### INI Definition

```
[ VCS ]  
Checkin = LMGOS2->LMGLBYCheckin
```

### Syntax

```
LONG _VMMAPI LMGLBYCheckin(HWND hwnd,  
                             char * type,  
                             char * subtype,  
                             char * filename);
```

Parameter	Description
Hwnd	An input parameter that is the parent/owner window handle.
type	An input parameter
Subtype	An input parameter.
Filename	An input parameter.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LMGLBYCheckout

## CheckOut

Called when user wants to get a writeable copy of a file, and lock it.

### INI Definition

```
[ VCS ]  
Checkout = LMGOS2->LMGLBYCheckout
```

### Syntax

```
LONG _VMMAPI LMGLBYCheckout(HWND hwnd,  
                             char * type,  
                             char * retfullname,  
                             char * path,  
                             char * ext,  
                             char * name,  
                             char * pszBtn,  
                             char * pszTitle);
```

Parameter	Description
hwnd	An input parameter that is the parent/owner window handle.
type	An input parameter. The list of files will be filtered based on the file type given. The type can be any LBY_TYPE_xxx define.
retfullname	An input/output parameter.  If "retfullname" is provided and is not empty, it will be used as a file name to extract the data into. If it is not provided or is an empty string, "path", "ext", and the actual name of the file selected will be used to put together a full path/file name for storing the record in.  If "retfullname" is provided, the resulting path/file name will be copied into it.  In other words, if you're supplying a temporary name in "retfullname", you could pass in "name" to find out what the actual name of the file was.
path	Optional input parameter.
ext	Optional input parameter.
name	An optional input/output parameter.  If "name" is provided, the name of the file will be copied into it.
pszBtn	An input parameter for the text on button.
pszTitle	An input parameter for the text in the title bar.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LMGLBYInit

## Init

This is called once, during program initiation.

## *INI Definition*

```
[ VCS ]  
Init = LMGOS2->LMGLBYInit
```

## *Syntax*

```
LONG _VMMAPI LMGLBYInit (void);
```

There are no parameters for this function

## *Remarks*

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LMGLBYReInit

## ReInit

Called whenever the master-resource settings have changed. Gives the VCS code a chance to reinitialize itself. (The LBY code needs to, because paths and file names for the LBY databases may have changed.)

### *INI Definition*

```
[ VCS ]  
ReInit = LMGOS2->LMGLBYReInit
```

### *Syntax*

```
LONG _VMMAPI LMGLBYReInit (void);
```

There are no parameters of this function

### *Remarks*

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LMGLBYSelect

## Select

### INI Definition

```
[ VCS ]  
Select = LMGOS2->LMGLBYSelect
```

### Syntax

```
LONG _VMMAPI LMGLBYSelect(HWND hwnd,  
                           char * type,  
                           char * retname,  
                           char * pszBtn,  
                           char * pszTitle,  
                           USHORT id,  
                           BOOL bAllVersions);
```

Parameter	Description
hwnd	An input parameter that is the parent/owner window handle.
type	An input parameter
Retname	An output parameter.
pszBtn	An input parameter for the text on the button.
pszTitle	An input parameter for text in the title bar
id	An input parameter for optional, ID of window
BAllVersions	An input parameter. True to show "all ver" btn

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.



---

# LMGLBYTerm

## Term

This is called once, during program term.

### *INI Definition*

```
[ VCS ]  
Term = LMGOS2->LMGLBYTerm
```

### *Syntax*

```
LONG _VMMAPI LMGLBYTerm (void);
```

There are no parameters for this function

### *Remarks*

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LMGLBYUnlock

## Unlock

Called when user wants to cancel changes to a file and release it.

### INI Definition

```
[ VCS ]  
Unlock = LMGOS2->LMGLBYUnlock
```

### Syntax

```
LONG _VMMAPI LMGLBYUnlock(HWND hwnd,  
                           char * type,  
                           char * subtype,  
                           char * filename);
```

Parameter	Description
Hwnd	An input parameter that is the parent/owner window handle.
type	An input parameter
Subtype	An input parameter.
Filename	An input parameter.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# LMGLBYView

## View

Called when user wants to select a file and get a read-only copy of it.

### INI Definition

```
[ VCS ]
View = LMGOS2->LMGLBYView
```

### Syntax

```
LONG _VMMAPI LMGLBYView(HWND hwnd,
                          char * type,
                          char * retfullname,
                          char * path,
                          char * ext,
                          char * name,
                          char * pszBtn,
                          char * pszTitle);
```

Parameter	Description
Hwnd	An input parameter that is the parent/owner window handle.
type	An input parameter. The list of files will be filtered based on the file type given. The type can be any LBY_TYPE_xxx define.
Retfullname	<p>An input/output parameter. If "retfullname" is provided and is not empty, it will be used as a file name to extract the data into. If it is not provided or is an empty string, "path", "ext", and the actual name of the file selected will be used to put together a full path/file name for storing the record in. If "retfullname" is provided, the resulting path/file name will be copied into it.</p> <p>In other words, if you're supplying a temporary name in "retfullname", you could pass in "name" to find out what the actual name of the file was.</p>
Path	Optional input parameter.
ext	Optional input parameter.
Name	An optional input/output parameter. If "name" is provided, the name of the file will be copied into it.
PszBtn	An input parameter for the text on the button.
PszTitle	An input parameter for text on the title bar.

### Remarks

This function returns SUCCESS (0) or FAIL (non-zero) to indicate the result.

---

# MajorVersion

Use this function to get the system's major version number.

*Syntax:*

MajorVersion ()

*Example:*

#MAJOR = MajorVersion ()

---

# MinorVersion

Use this function to get the system's minor version number.

*Syntax:*

```
MinorVersion ()
```

*Example:*

```
VERS = MajorVersion() & '.' & MinorVersion()
```

---

# PrinterClass

Use this function to determine the type of print stream being generated.

*Syntax:*

```
PrinterClass()
```

*Example:*

```
IF (PrinterClass() = 'AFP') THEN  
  AddComment(AppldxRec() )  
END
```

---

# PrinterGroup

Use this function to determine the group name in the INI file being used to generate the print stream.

*Syntax:*

PrinterGroup()

*Example:*

ScriptName = GETINISTRING(, PrinterGroup(), 'DocuSaveScript')

---

# PRINT\_IT

Use this function to print a string to the console.

*Syntax:*

PRINT\_IT (Text)

Parameter	Description	Required?
Text	String to be printed to the console	Yes

*Example:*

Here is an example:

```
IF (HaveGVM('Company'))  
  PRINT_IT( GVM('Company') )  
END
```



---

# SetGVM

Use this function to update the contents of a GVM variable. You can also use this function to create a GVM variable

## *Syntax:*

SetGVM (VarName, VarData, Instance, VarType, VarSize)

Parameter	Description	Required?
VarName	String containing the name of the GVM variable	Yes
VarData	Data to store in the GVM variable	Yes
Instance	Instance number of the GVM variable (default = 1)	No
VarType	Type of GVM variable to create: 'C' = Character array 'S' = Short 'L' = Long 'F' = Float 'D' = Double 'Q' = Long Double	No
VarSize	Number of bytes to reserve when creating GVM variable	No

## *Example:*

Here are some examples:

```
IF (HaveGVM('Company'))
  SetGVM('Company', 'My Company' )
END
IF (HaveGVM('My Variable') = 0)
  SetGVM('My Variable', 'My Data', 'C', 50)
END
```

---

# TMRTimers

Programs that use a .RES file for menu definition can start and stop “automatic” functions by adding a menu item to the .RES file, identifying the entry point function of TMRLIB. This option could be chosen manually by the user or enable automatically with the “Startup” menu ID defined in the INI file. Any menu ID value designated in the INI executes when the program begins.

To add this option to an existing menu system, edit the proper “menu”.RES file and add the following line:

```
MENUITEM  "Timed functions",  2001  "TMROS2->TMRTimers"  "Start/Stop auto-  
timer"
```

The menu item and description strings can say whatever you want. The menu ID must not conflict with any other IDs.

Placement of this menu item among the other items is not important. However, you probably place it “out of the way” since the users will probably not need access to the option very frequently.

By default, TMRLIB will attempt to place a checkmark next to the menu item when the timer is activated. Then checkmark will be removed when the timer is stopped.

Documaker Workstation automatically starts and stops timer functions beginning with version 9.0, therefore this menu option will not be required in that environment.

---

# TMRInit

This function attempts to initialize and all registered service functions. Each DLL is loaded and the service functions queried. Those that load successfully will receive the message FAP\_MSGINIT. Parameters to this function provide the application HAB (or HINSTANCE under Windows); the handle to the main application windows; and a pointer to any application specific data that needs to be passed to all service functions. *Under PPS/Entry, the data pointer will address the AFEDATA structure.*

The function returns SUCCESS (0) if initializations complete without error.

---

# TMRTerm

This function will send the FAP\_MSGTERM message to all service functions. The OS timer is disabled and then released.

After this call, service functions will no longer be called.

---

# TMRSetAppData

This function will store the application specific data pointer for use by service functions. Under Documaker Workstation, this pointer must reference the AFEDATA structure.

---

# TMRAppData

This function will return the application specific data pointer for use by service functions. Under Documaker Workstation, this pointer must reference the AFEDATA structure.

---

# TMRSetHwnd

This function will store the application's main window handle for use by service functions.

---

# TMRHwnd

This function will return the application's main window handle for use by service functions.



---

# TMRSetHab

This function will store the application HAB (or HINSTANCE for Windows) for use by service functions.

---

# TMRHab

This function will return the application HAB (or HINSTANCE for Windows) for use by service functions.

---

# TMRIsDesktopUp

FAPFormset() is used to retrieve the current form set. If there is a form set and a form within that form set has a FAPWINDOW structure associated with it, this function will return TRUE. Otherwise, the function returns FALSE.

---

# TMRIsDialogUp

All child windows associated with the application main window (returned by TMRHwnd()) will be scanned to determine if a window is present. If a window is found, TRUE is returned. Otherwise, the function returns FALSE.

The existence of a window usually means that the user is involved with data entry or some other functionality.

---

# TMRTimerTest1

This is a test function and may be used by any application to test the timer. The function will display a window that includes information about internal settings.

---

# TMRTimerTest2

This is a test function and may be used by any application to test the timer. The function will display a window that includes information about internal settings.

---

# TMRTimerTest3

This is a test function and may be used by any application to test the timer. The function will display a window that includes information about internal settings.

---

# TRNAutoKeyIDUsrFunc

## AutoKeyID

This hook is called to provide, verify, or release KeyIDs for WIP transactions. This optional feature must be enabled via INI settings. This feature first became available in July 1996.

Custom versions of this function can return any value that satisfies the KeyID requirements.

The standard function, TRNAutoKeyIDUsrFunc(), can be used to return sequential numbers generated in a table via another function.

### INI Definition

```
[ AfeProcedures ]
AutoKeyID = TRNOS2->TRNAutoKeyIDUsrFunc
```

### Syntax

This function must conform to the FAPUSER prototype. For more information on the FAPUSER prototype, see the [FAPUSER](#) section. You can use the following parameters:

Parameter	Description
DwMessage	<p>A message requesting a particular operation. The following FAP messages are passed in the dwMessage parameter for the following operations, and should be handled accordingly in a custom procedure. Please note that although FAP message numbers are being used, there is no FAPOBJECT that initiates or receives the action.</p> <p><b>FAP_MSGNEXT</b> Get a new unused KeyID if the current one is not valid. This message is sent any time the Form Selection window is (re)initialized.</p> <p><b>FAP_MSGUPDATE</b> Release the KeyID. Its use was aborted. This message is sent if the Form Selection window is canceled while creating a new form set or the form set is not saved to WIP. This message will be sent if the WIP is deleted without archiving. This message will also be sent any time the user manually changes the KeyID. This might occur if WIP is edited and the KeyID is changed by user action.</p> <p><b>FAP_MSGDELETE</b> Form set was archived. KeyID may be deleted. This message will only be sent when the form set has been archived.</p> <p><b>FAP_MSGSELECT</b> Verify that the current KeyID is valid for situation. This message is sent when the OK action is taken on the Form Selection window. This message is also sent any time the Transaction Code, Key1, or Key2 selections change. This should make it possible for a custom function to generate specific Ids based upon this information (if desired).</p>
DwFAPHab	The program's anchor block or instance handle. The distinction depends upon whether the program is running on an or Windows platform. Within the Docucorp programming environment, both definitions serve the same purpose.
DwFAPHwnd	A handle to the currently open window. You should not assume a particular window handle is being passed because this function can be called from multiple locations



	within AFE.
dwChildID	The ID of a child control on the window that initiated the call or zero if no child ID is available.
DwObjectType	Not used.
DwInputFlag1	Not used.
DwInputFlag2	Not used.
DwInputFlag3	Not used.
LpszUserID	A pointer to a NULL terminated string that contains the user ID of the current operator.
LpszTranCode	A pointer to a NULL terminated string that contains the Transaction Code associated with the WIP record.
lpszKey1	A pointer to a NULL terminated string that contains the Key1 field value associated with the WIP record.
lpszKey2	A pointer to a NULL terminated string that contains the Key2 field value associated with the WIP record. Note that in a multi-select situation (PPS) only the first Key2 value is provided.
LpszInputBuffer	A pointer to a NULL terminated string that contains the current KeyID field value associated with the WIP record.
LpszKeyID	A pointer to a text buffer that should receive the output KeyID from this function. If this value is NULL, no output string is expected. On input, a non-NULL value will represent the last KeyID returned from your hook procedure. A difference between the input KeyID and the one represented in this string means that the user changed the original KeyID.
DwOutputBufferMaxSize	Represents the maximum size of the output buffer for the previous parameter.
pAFEData	A pointer the current AFEDATA structure used by AFE. This structure contains most (if not all) of the global data necessary to manipulate the form sets in use by the Entry module. Even if you are familiar with AFELIB and how the Entry module works, you should use care when manipulating the data within this structure.
LpdwOutputFlag2	Not used.
LpdwOutputFlag3	Not used.

### Remarks

This function must return SUCCESS (0) or FAIL (non-zero) to indicate whether the operation succeeded. Upon a successful return, the value contained in *char FAR \* lpszOutputBuffer* (if not NULL) will be copied as the new KeyID of choice.

The parameters provided to this function includes all the current WIP information required to understand how the user constructed (or is constructing) the form set. A custom function may or may not use the information provided.

---

# TRNSetBannerFormInfo

## Set Banner Information

If specified, a banner page is printed for every recipient in each form set. No default INI option definition is assumed, therefore one must be provided to enable the functionality.

The standard function, TRNSetBannerFormInfo(), is available for use and uses additional INI groups and options to determine what is printed on the banner page.

In addition to filling the banner page, the standard function will search the remaining form set and fill in the field information on any PULL forms defined. A PULL form is a representation that an externally maintained form should be included in the form set. When the form set is printed, a separate page is printed for each PULL form and usually indicates what form should be replacing that page.

### INI Definition

```
[ AFEProcedures ]  
BannerProc = TRNOS2->TRNSetBannerFormInfo
```

### Syntax

This function must conform to the FAPHANDLER prototype. For more information on the FAPHANDLER prototype, see the [FAPHANDLER](#) section. You can use the following parameters:

Parameter	Description
<i>ObjectH</i>	The handle of the form that represents the banner page.
<i>msgno</i>	Message FAP_MSGINIT. Receiving this message means that all banner information should be applied to the form handle passed.
p1	This parameter is passed as NULL and should not be used.
p2	This parameter is passed as NULL and should not be used.

### Remarks

This function should return SUCCESS (0) or FAIL (non-zero) to indicate the result of the operation.

Any information from any source may be placed on the banner form. Note that the PULL form functionality should be performed if compatibility is to be maintained.