# DocMe

## User's Guide

# Contents

# 1. Introduction to DocMe

## 1.1 Overview

### 1.1.1 Summary

DocMe provides a system for the rapid production of correct documentation for computer code. The software developer inserts comments and explanations within the source files bracketed by DocMe format codes. The DocMe program accepts the source code file and converts it into an .rtf document, a Microsoft Help File and an HTML help file. The .rtf file can be opened in Microsoft Word™ and assigned a DocMe template which formats a finished document.

DocMe provides the following benefits to software developers:

- Code and documentation are maintained in the same files for optimum safety and simplicity of maintenance.
- Changes to code and documentation are made in the same place, resulting in consistent and timely upgrades to documentation.
- The documentation exactly matches the documentation of the source code, which reduces frustrating situations for users.
- The time to produced parallel documentation copy, Microsoft Help files, and HTML Help files is reduced to minutes. This yields dramatic cost reductions and allows frequent production of updated documentation.
- In line documentation makes programs easy to understand for later debugging, maintenance and upgrades.

### 1.1.2 Features and Functions

DocMe provides the following features and functions:

- DocMe provides a variety of headings and subheading items appropriate for organizing software code.
- Each heading is associated with a collection of required, optional, and unavailable subheading items. The restrictions enforce a consistent documentation style.
- DocMe extracts blocks of text based on block begin and end codes. These codes allow the extraction of text to be seen by users and text to be seen in-house.
- DocMe imbedded codes set bold, underlined, or italicized font styles.
- DocMe interprets embedded pointer codes to automatically create hot button jumps to definitions of entities in the Word document, Microsoft Help file and HTML Help file.
- DocMe allows a list of files to be processed together into a single document. The list may be arranged into outline form with multiple heading levels for conceptual organization.
- DocMe allows the entities of the same type to be automatically listed alphabetically or left in edited order.

### 1.1.3 Top Level Architecture

Formatting codes and keywords are inserted in source code to direct the operation of the DocMe compiler. A list of the file names is created and a copy is sent to the **DocMe.exe** program. The program creates four files, a cross reference file, a Microsoft Word .rtf file, a help reference file, and an HTML Help file. The Word File may be assigned a template and edited. the help reference file is sent to the Microsoft help compiler program hcrtf.exe which creates a Microsoft Help file. The following figure illustrates the sequence of the process.

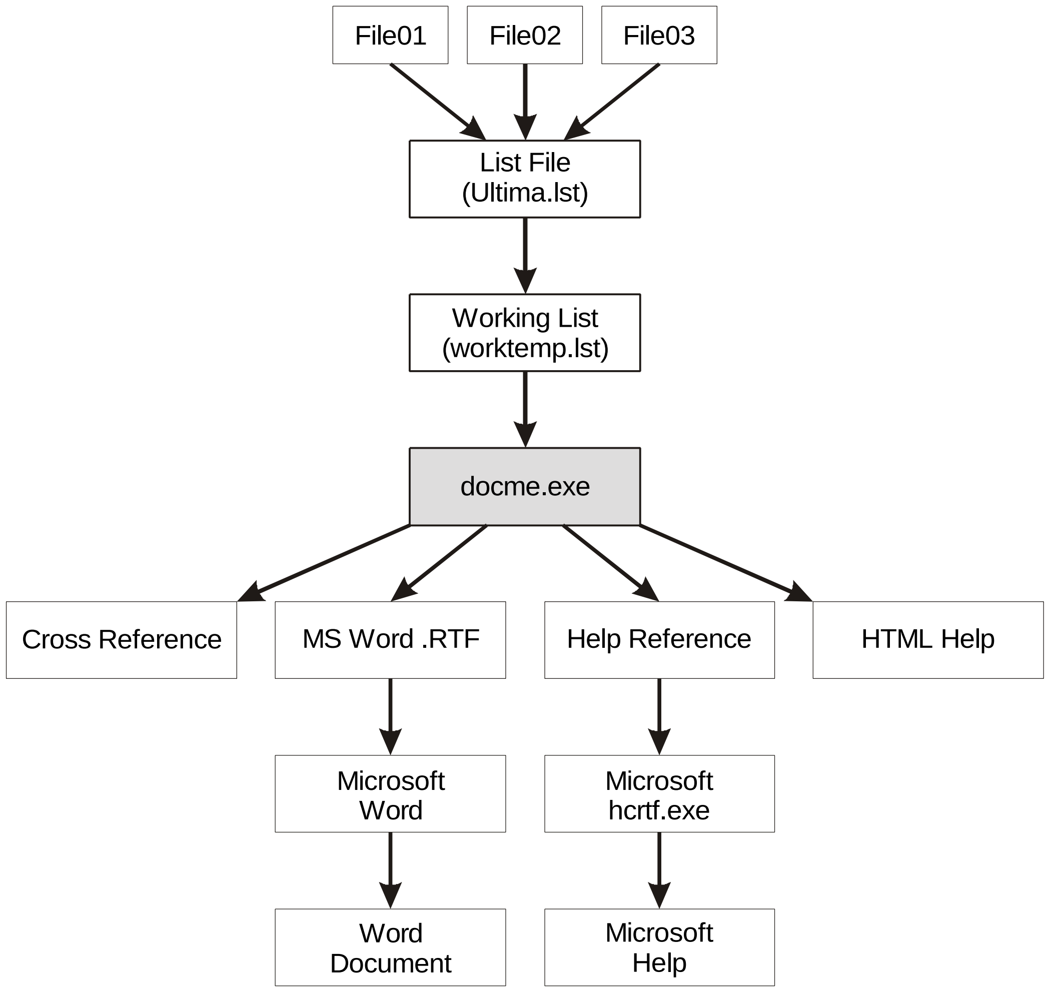

**Figure 1 - The DocMe processing sequence**

# 2. Installing DocMe

Please follow this simple few steps for installation:

- Unzip the library into a directory named DocMe.
- Copy  Microsoft help compiler HCW.EXE, RTF compiler HCRTF.EXE and the support DLL, HWDLL.DLL to the BIN directory  under the DocMe directory
- You are ready to go.

***Note***

The recommended way to use the product is to include it is the post build action. This action will generate a fresh documentation from the source code.

# 3. Support

For support questions recommendations and requests, please contact Pickle Software via:
Koby@msn.com

# 4. Creating Files for DocMe

DocMe documents consist of sections which define entities. Each entity has an item type and a name. Subsections describe the properties of the entity.

## 4.1 Item Types

Each item heading is indicated by a key word, followed by an equal sign, followed by the name of the item. For example, a class named Control is introduced by the expression:

**Class = Control**

DocMe will later create a heading **Class**. The sub-heading **Name** will be automatically added with the name **Control** inserted.

The following item types are available:

| | |
|---|---|
| **Class** | C++ class |
| **C++ Method** | C++ methods |
| **Constant** | constants, #defined constants, variables declared as constants |
| **Data type** | data types — structures, enumerated types, typedefs, etc. |
| **Function** | functions |
| **Function Type** | data types that are pointers to functions |
| **Macro** | macros |
| **Module** | modules |
| **Object** | object descriptions |
| **Term** | special term which requires definition |
| **Variable** | variables |

## 4.2 Sub-headings

Each item type heading is followed by a number of subheadings. Some subheadings are required for the item type, some sub-headings are optional, some sub-headings are unavailable.

A sub-heading is introduced by by a colon (**:**) prefix followed by an appropriate keyword. For example a **Class** entity may have an **:overview** sub-heading, a **:description** sub-heading, and a **:see also** sub-heading, as shown below:

**Class = MMMControl**

**:overview**

This is the overview text.

**:description**

This is the description text.

**:see also**

This text describes other references.

The following table shows all sub-headings and the item types to which they apply. The headings are either required (R), optional (O), or not available (Blank) for each of the item types:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Subsection Headings** | Meth | Class | Const | Type | Func | FncTyp | Macr0 | Note | Objct | Term | Var |
| **Overview** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** |
| **Syntax** | | | **R** | **R** | **R** | **R** | O | | | | **R** |
| **PRototype-In** | **R** | **R** | | | **R** | **R** | O | | O | O | |
| **Prototype** | **R** | | | **R** | **R** | **R** | | | | | |
| **Defined-In** | O | | **R** | **R** | O | O | O | | | | **R** |
| **Files** | | | | | | | | | O | O | O |
| **Preconditions** | O | | O | O | O | O | | | | O | O |
| **Parameters** | **R** | | O | O | O | O | **R** | | | | O |
| **Fields** | | | O | O | | | | | | | O |
| **Elements** | | | O | O | | | | | | | O |
| **Tokens** | | | | | | | | | | | |
| **Events** | | | | | | | | | | | |
| **Event-Data** | | | | | | | | | | | |
| **Confirm-Data** | | | | | | | | | | | |
| **Attributes** | O | | O | O | O | O | O | | O | O | O |
| **Data Types** | O | | O | O | O | O | O | | O | O | O |
| **Methods** | O | | O | O | O | O | O | | O | O | O |
| **Description** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | **R** |
| **Assumptions** | O | | O | O | O | O | O | | O | O | O |
| **Used-By** | O | | O | O | O | O | O | | O | O | O |
| **Remarks** | O | O | O | O | O | O | O | O | O | O | O |
| **Limitations** | O | | O | O | O | O | O | | O | O | O |
| **Return Value** | O | O | O | O | O | O | | | | | O |
| **Portability** | O | O | O | O | O | O | O | | O | O | O |
| **Example** | **R** | O | O | O | **R** | **R** | **R** | | O | O | O |
| **Data Members** | | O | | | | | | | | | |
| **Creation type** | | O | | | | | | | | | |
| **See Also** | **R** | **R** | **R** | **R** | **R** | **R** | **R** | O | **R** | **R** | **R** |

## 4.3 Text

Explanatory text may be inserted below the sub-headings. Formatting codes may be imbedded in the text to create bulleted lists, special font characteristics, or to indicate a pointer.

DocMe will automatically add Word styles for the headings and text.

### 4.3.1 Text Formatting Commands

The following formatting codes can be imbedded in the text:

| | |
|---|---|
| **@** | bulleted, indented paragraph |
| **/*+**<text>**+*/** | text selected for customer documentation output. |
| /#<text>#/ | used instead of **/*** and ***/** for comments within the DocMe Item Description. The C compiler interprets ***/** as a signal to end the Description (**/*+** and **+*/**) instead of the comment within the description. The source code will be read incorrectly by the C ’compiler and, in most cases, this will cause fatal errors in the compilation! |
| **\|**<text> | set text line with mono font |
| **^x**<text>**^n** | link to definition, mono font |
| **^l**<text>**^n** | set text with mono font all words that relate to code, such as function names, variables, parameter names, constants, file names, directory listings, etc., |
| **^e**<text>**^n** | set text with italics |
| **^b**<text>**^n** | set text with bold |

| | |
|---|---|
| **^u**<text>**^n** | set text with underline. |

## Notes

The font styles may be mixed. For example, **^b^u**<text>**^n** yields bold underlined text.

The codes may be expressed in either upper case or lower case.

Single spaces which are inserted between the **`^L`**, <text>, and **`^N`** will be ignored by the DocMe utility. DocMe ignores any spaces at the end of a text line. If there is more than one sentence in a text block, avoid ending a sentence at the end of a line. The resulting text will have no spaces between the two sentences.

Do not use **`^L, ^N, |`**, or other special formatting characters in the **:overview** or **:see also** sections. The DocMe utility formats these sections.

# 5. Sample Document

The following is a sample of API text formatted for the DocMe utility. The DocMe formatted .rtf output is shown after the text file

## 5.1 Sample API Text

```
/*+-----------------------------------------------------------------
Class = MMMControl

:Overview
Initialize and do message dispatching for user applications.

:Prototype In
| mmm.h

:Description
An ^LMMMControl^N object initializes user applications with the
^XMMMInitialize^N method. The objects provide message queuing, and
message dispatching, for both repeater applications and user
applications. The ^LMMMControl^N object creates a message queue for
its applications and dispatches messages to the application callback
function, if one is specified.

:See Also
^XMMMInitialize^N

----------------------------------------------------------------- +*/


/*+-----------------------------------------------------------------
Data Type  = MMMAddress

:Overview
  A structure containing a transport protocol and an address.

:Syntax
  |typedef struct
  |{
  |   unsigned short        transport_identifier;
  |   MMMTransportAddress   transport_address;
  |} MMMAddress;

:Defined In
  |mmm.h

:Description
  The data type ^LMMMAddress^n is a structure that is used to
specify a transport stack protocol and a network address for a
network repeater.
```

**:Fields**
  **^L**transport_identifier **^N** identifies the transport stack protocol. The valid values are:
  @  **^L**MMM_TRANSPORT_TCP **^N** -transport control protocol, for both user and repeater communications
  @  **^L**MMM_TRANSPORT_RMTP **^N** - reliable multicast transport protocol, for user communication
  @  **^L**MMM_TRANSPORT_RMP **^N** - reliable multicast protocol, for repeater communication
**^L**transport_address **^N** is a NULL terminated string specifying a network address in the form  hostname:port.

**:See Also**
MMMTransportAddress

-------------------------------------------------------------------  **+*/**

**/*+**-------------------------------------------------------------------

**C++ Method =** MMMInitialize

**:Overview**
  This method is used to initialize a remote user application.

**:Prototype**
```
  |MMMError MMMInitialize
  |(
  |   MMMVersion       mmm_version_requested,         // INPUT
  |   MMMAddress       mmm_repeater_address,         // INPUT
  |   MMMVersion      * mmm_version,                  // OUTPUT
  |);
```

**:Prototype In**
```
  |mmm.h
```

**:Description**
The C++ method **^L**MMMInitialize() **^N** initializes the remote user's **^X**MMMControl**^N** object and returns the negotiated version of MMM.

**:Parameters**
  **^L**mmm_version_requested **^N** - the version of the MMM API which the application requires
  **^L**mmm_repeater_address **^N** - an **^X**MMMAddress**^N** structure containing the address of the repeater server
  **^L**mmm_version **^N** - a pointer to a structure containing the version that will be used

**:Return Value**
  @  If successful, this method returns the value **^L**MMMERROR_NO_ERROR **^N**. A non-zero return indicates the specific failure condition.
  @  **^L**MMMERROR_ALREADY_REGISTERED **^N** - The application has already been initialized by the MMM system.
  @  **^L**MMMERROR_GLOBAL_MANAGER_NOT_RESPONDING **^N** - The global manager is not responding to requests.

  @  **^L**MMMERROR_INVALID_PARAMETER **^N** - One of the parameters to the request is invalid.
  @  **^L**MMMERROR_NO_ERROR **^N** - The function call was successful.
  @  **^L**MMMERROR_NO_REPEATER_PROFILE **^N** - The repeater profile is not available.

**:Example**
  **- REQUIRED -**
  The following example shows the method code in use....

**:See Also**
**^X**MMMControl**^N**

----------------------------------------------------------------- **+*/**

## 5.2 DocMe Output

The following shows the DocMe formatted .rtf output. The headings have been modified for insertion into this document.

### 5.2.1 MMMControl **worktemp.txt 2 Class**

#### Names

MMMControl

#### Overview

Initialize and do message dispatching for user applications.

#### Prototype In

`mmm.h`

#### Description

An `MMMControl` object initializes user applications with the `MMMInitialize` method. The objects provide message queuing, and message dispatching, for both repeater applications and user applications. The `MMMControl` object creates a message queue for its applications and dispatches messages to the application callback function, if one is specified.

#### See Also

`MMMInitialize`

### Names

MMMAddress

### Overview

A structure containing a transport protocol and an address.

### Syntax

```
typedef struct
{
  unsigned short        transport_identifier;
  MMMTransportAddress   transport_address;
} MMMAddress;
```

### Defined In

`mmm.h`

### Description

The data type `MMMAddress` is a structure that is used to specify a transport stack protocol and a network address for a network repeater.

### Fields

`transport_identifier` identifies the transport stack protocol. The valid values are:

- `MMM_TRANSPORT_TCP` -transport control protocol, for both user and repeater communications
- `MMM_TRANSPORT_RMTP` - reliable multicast transport protocol, for user communication
- `MMM_TRANSPORT_RMP` - reliable multicast protocol, for repeater communication

`transport_address` is a NULL terminated string specifying a network address in the form hostname:port.

### See Also

MMMTransportAddress

## Names

MMMInitialize

## Overview

This method is used to initialize a remote user application.

## Prototype

```
MMMError MMMInitialize
(
  MMMVersion       mmm_version_requested,         // INPUT
  MMMAddress       mmm_repeater_address,         // INPUT
  MMMVersion       * mmm_version,                 // OUTPUT
);
```

## Prototype In

`mmm.h`

## Description

The C++ method `MMMInitialize()` initializes the remote user's `MMMControl` object and returns the negotiated version of MMM.

## Parameters

`mmm_version_requested` - the version of the MMM API which the application requires `mmm_repeater_address` - an `MMMAddress` structure containing the address of the repeater server `mmm_version` - a pointer to a structure containing the version that will be used

## Return Value

- If successful, this method returns the value `MMMERROR_NO_ERROR`. A non-zero return indicates the specific failure condition.
- `MMMERROR_ALREADY_REGISTERED` - The application has already been initialized by the MMM system.
- `MMMERROR_GLOBAL_MANAGER_NOT_RESPONDING` - The global manager is not responding to requests.
- `MMMERROR_INVALID_PARAMETER` - One of the parameters to the request is invalid.
- `MMMERROR_NO_ERROR` - The function call was successful.
- `MMMERROR_NO_REPEATER_PROFILE` - The repeater profile is not available.

## Example

- REQUIRED - The following example shows the method code in use....

## See Also

`MMMControl`

## 5.3 Help Files

The DocMe utility automatically creates a Microsoft Help file and an HTML Help file. The files are formatted and hot buttons are added at the words and phrases indicated by imbedded codes. The worktemp.lst file (filled from Ultima.lst) is used to create the table of contents for the help files. The initial portion of the HTML Help file for the source file is shown below:

### 5.3.1 HTML Help File

MyApi

Table of Contents

- MyApi
- Product API

*Generated: Fri Sep 25 09:20:56 1998*

---

Sections

**Product API**

| | | |
|---|---|---|
| MMMControl | Class | Initialize and do message dispatching for user applications. |
| MMMAddress | Data Type | A structure containing a transport protocol and an address. |
| MMMInitialize | C++ Method | This method is used to initialize a remote user application. |

---

**MMMControl** worktemp.txt:2 **Class**

---

**Names:**

MMMControl

**Overview:**

Initialize and do message dispatching for user applications.

**Prototype In:**

`mmm.h`

**Description:**

An `MMMControl` object initializes user applications with the `MMMInitialize` method. The objects provide message queuing, and message dispatching, for both repeater applications and user applications. The `MMMControl` object creates a message queue for its applications and dispatches messages to the application callback function, if one is specified.

**See Also:**

`MMMInitialize`

## 5.4 Word Documents

The DocMe.dot template may be used to format the .rtf file as a Word file. This template creates an index based on the item type headings.

## 5.5 Cross Reference File

The output cross reference file, worktemp.xref in this example, shows references to and from each items. Here is the cross reference file from the source file above:

```
 ------------- DocMe - Cross Reference -------------

  1 - MMMControl
      refers to...
           3 - MMMInitialize  ( MMMInitialize)
           3 - MMMInitialize  ( MMMInitialize)
      referred by...
           3 - MMMInitialize
           3 - MMMInitialize

  2 - MMMAddress
      refers to...
      referred by...
           3 - MMMInitialize

  3 - MMMInitialize
      refers to...
           1 - MMMControl  ( MMMControl)
           2 - MMMAddress  ( MMMAddress)
           1 - MMMControl  ( MMMControl)
      referred by...
           1 - MMMControl
           1 - MMMControl
```

## 5.6 Source Code Content Rules

Source code documentation is converted to the On-Line help and Reference Manuals for each product.  Attention to detail pays-off in the creation of more usable documentation for your customers.

- Source files must contain all required reference documentation items.
- No customer exposed data types should be contained in a .c file.
- No function should be described in an .h file.
- Item descriptions for functions, macros, etc.,  should be contained in the .c file that contains the definition of that item.
- Item descriptions for data types, function types (pointers to functions), constants, etc., should be contained in the .h files that contain the definition of that item.
- All other related documentation should be included in the Developer's Guide or Programming Manual of that product.

## 5.7 Spaces and Carriage Returns

DocMe ignores spaces and carriage returns when formatting. The example below shows text with arbitrary carriage returns and spaces. The .rtf output from DocMe will be formatted without the extraneous carriage returns and spaces..

```
:Description
The MMM dictionary command  ^L PortAssociate^N creates an
association between
the
two ports represented by the tokens  ^L PortID1^N and ^L PortID2^N.
This command
first translates the two tokens from the  ^L .BTC^N file into the
appropriate
parameters that are required for making the association between
the two ports,
and then it performs the association.
```

Here is the DocMe formatted version.

**Description**

The MMM dictionary command `PortAssociate` creates an association between the two ports represented by the tokens `PortID1` and `PortID2`. This command first translates the two tokens from the `.BTC` file into the appropriate parameters that are required for making the association between the two ports, and then it performs the association.

## 5.8 Creating Lists

Lists require the use of bullets, or special formatting. The following code will **not** produce a numbered list:

```
:Description
  The function takes the following values:
   1) 60A349h
   2) 60B349h
   3) 60C349h
   4) 60D349h
```

DocMe produces the following output:

**Description**

The function takes the following values: 1) 60A349h 2) 60B349h 3) 60C349h 4) 60D349h

The DocMe compiler ignores spaces and carriage returns in order to create a clean document from a text based source file. To create a list, use the bulleted list delimiter '`@`', or use the `'|'` character on a line between each list item. The following examples illustrate the techniques.

```
:Description
  The function takes the following values:
   @ 60A349h
   @ 60B349h
   @ 60C349h
   @ 60D349h
```

or:

```
:Description
The function takes the following values:
   |
   1) 60A349h
   |
   2) 60B349h
   |
   3) 60C349h
   |
   4) 60D349h
```

## 5.9 Templates for Formatting Source Code

Templates are available for each of the different item types in the file **`template.c`**.

# 6. Running DocMe

## 6.1 Setting Up the DocMe Files

The following sub-directory structure illustrates one method of running DocMe:

| | |
|---|---|
| \DocMe | |
| DocMe.exe | the DocMe program |
| make.bat | a batch file to control DocMe |
| ultima.lst | the primary list of work files for DocMe to process |
| \input | |
| worktemp.txt | a work file to process |
| worktemp2.h | another work file to process |
| ... | |
| \output | the DocMe output files will go here |

In production settings the files can also be left in place, rather than using the \input directory. The make.bat file should be changed accordingly.

## 6.2 The Program DocMe.exe

### 6.2.1 Parameters and Options

The program **DocMe.exe** accepts a number of parameters which determine its output and operates on the files listed in the file named in the last parameter.

**DocMe [options] [<list_file_path>]**

options:

| | |
|---|---|
| **-d** | print debug info |
| **-g w4w** | generate a Word for Windows .rtf file |
| **-g winhelp** | generate a Windows Help reference .htf file |
| **-g html** | generate an .html Help file |
| **-g xref** | generate a cross reference file |
| **-h** | print this usage help message |
| **-p** | include proof reading information |
| **-q** | suppress diagnostic messages |
| **-s** | sort alphabetically |
| **-t <title>** | set document title |
| **-S** | print valid entry sections |

For example,

```
DocMe.exe -g xref -g winhelp -g w4w -g html -p worktemp.lst
```

The file, **worktemp.**lst in this example, contains a list of one or more files to be processed. The name, **worktemp**, of this file determines the names of all the output files. The **DocMe.exe** program creates the following files:

| | |
|---|---|
| **worktemp.xref** | a cross reference file |
| **worktemp.htf** | a Microsoft Help reference file, used to make the Help file |
| **worktemp.rtf** | a rich text format file to be formatted as a document |
| **worktemp.html** | an html help file |

The file **Ultima.lst** will be copied to create **worktemp.lst**. This allows the **worktemp** files to be cleared while compiling intermediate versions of a document without losing the list file contents. The **make.bat** file will handle clearing and copying.

## 6.3 The List File

The file **worktemp.lst** (copied by **make.bat** from the original version stored in **Ultima.lst**) contains a list of files to be processed, organized into an outline structure. Lines prefixed with '$', '$$', or '$$$' are arbitrary outline headings chosen by the author. The '$$' prefix indicates an indented line. Here is a sample list file. In this case only a single file **input\worktemp.txt** will be processed.

```
$MyApi
    $$Product API
            input\worktemp.txt
```

The list of files may be extensive and may be organized in more detail, as the following example illustrates:

```
$RPKernel
    $$RPKernel
            $$$Tasks
                    X:\kernel\new\ker_tsk.c
                    X:\kernel\new\ker_scd.c
            $$$Semaphores
                    X:\kernel\new\ker_sem.c
            $$$Messages/Queues
                    X:\kernel\new\ker_msg.c
            $$$Tokens
                    X:\kernel\new\ker_tkn.c
            $$$Timers
                    X:\kernel\new\ker_utm.c
            $$$General
                    X:\kernel\new\ker_hdl.c
                    X:\kernel\new\ker_man.c
                    X:\kernel\new\kernel.d
                    X:\kernel\new\ker_cpu._d
                    X:\kernel\new\ker_asn.c
                    X:\kernel\new\ker_err.c
                    X:\kernel\new\ker_rtc.c
```

```
                X:\kernel\new\ker_vct._d
                X:\kernel\new\ker_dgn.c
                X:\kernel\new\kdbg_xx.d
        $$$Dos Manager
                X:\kernel\new\386\dos_man.c
                X:\kernel\new\386\kerdos.d
```

The $, $$ and $$$ indicators are used  to create titles and the user menus for the Microsoft Help file and the HTML Help file.

## 6.4 Creating a Microsoft Help File

In the current example, DocMe creates a Microsoft Help reference file **worktemp.htf**. The batch file **make.bat** causes this file to be processed by the Microsoft Help program **hcrft.exe** to create a Microsoft Help file **worktemp.hlp**. In order to create a Microsoft Help file, the Microsoft Help Workshop files **, hcrft.exe, hcw.exe** and **hwdll.dll** must be on the path or in the DocMe directory.

The command,

**hcrtf -o worktemp.hlp -xr worktemp.htf**

generates the Microsoft Window Help file, **worktemp.hlp**, from the reference file **worktemp.htf**.

## 6.5 The make.bat File

The batch file, **make.bat**, automates the job process. The contents of **worktemp.lst** containing the list of files to be processed will be stored in a file **ultima.lst** for reference and copied to **worktemp.lst** when required. This allows files to be **worktemp.*** files to be erased without loosing the list of files. The batch file will clear out old versions, **worktemp.***, copy **ultima.lst** to **worktemp.lst**, run **DocMe.exe**, create the Windows Help file, move the files to the **\output** directory, and start the help files for checking.

The batch file, **make.bat**, looks like this:

```
Rem - Delete old output files from the output directory.
@for %%f in (worktemp.* output\*.*) do del %%f /q

Rem - copy the configuration file ultima.lst to the temporary file worktemp.lst.
copy ultima.lst worktemp.lst

Rem - Run DocMe.exe using info in worktemp.lst.
DocMe.exe -g xref -g winhelp -g w4w -g html -p  worktemp.lst

Rem - Run the MS Help compiler to create file worktemp.hlp using file worktemp.htf.
hcrtf -o worktemp.hlp -xr worktemp.htf

Rem - Move all files created to the output directory.
@for %%f in (worktemp.*) do move %%f output
```

Rem - Start the MS Help and the HTML Help to check them out.

**start output\worktemp.hlp**

**start output\worktemp.html**

**:end**

## 6.6 Creating a Word Document

The template file **DocMe.dot** should previously be copied into the root of your Word for Windows directory (or your assigned Word for Windows Templates directory)

- ☐ Start Word for Windows.
- ☐ Select **File/New**
- ☐ Select **DocMe.dot** as your template.

Upon entering the file, you will be prompted to enter the **File/Properties/Summary Information**.

- ☐ Enter the Title column only, the rest of the information is already setup.  Press OK.

Load the .rtf documentation file as follows:

- ☐ Press the Yellow Smiley Icon on the top tool bar.
- ☐ Select **Load** new documentation (.rtf) file.
- ☐ Select OK to delete all existing DocMe example documentation before loading the new documentation file.
- ☐ Select the .rtf file to be processed. Press OK and allow conversion.
- ☐ When prompted to update, press OK. DocMe will format the document.
- ☐ Save the document in Word format.

# 7. Item Prototypes

This section provides examples of the DocMe item types, along with a full list of required and typical optional entries for each item. Each entry field is described, shown in the proper order, and in some cases, examples are provided.

## 7.1 Class = <C++ Class Name>

**Overview**

**[Required]** A short description of the C++ class 'functionality. This is used as the heading for locating the item in the On-Line help systems. In order to make the Windows Help menu system easy to navigate and simple to use, do not make your Overview entries longer than 65 characters.

**Prototype In**

**[Required]** The name of the .HPP file that contains the prototype of the C++ Method.

**Description**

**[Required]** This is a more detailed description of the item's function than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that.

**Remark**

[Optional]

**Return Value**

[Optional]

**Portability**

[Optional]

**Example**

[Optional]

**See Also**

**[Required]**

## 7.2 C++ Method = <C++ Method Name>

**Overview**

**[Required]** A short description of the C++ Method's functionality. This is used as the heading for locating the item in the On-Line help systems. In order to make the Windows Help menu system easy to navigate and simple to use, do not make your Overview entries longer than 65 characters.

**Prototype**

**[Required]** A complete listing of the C++ Method.

```
reactivate( DFEState* state)
```

**Prototype In**

**[Required]** The name of the .HPP file that contains the prototype of the C++ Method.

**Description**

**[Required]** This is a more detailed description of the item's function than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that. The description for C++ Methods are for internal use only.

**Preconditions**

[Optional] If applicable, one or more possible preconditions that are required by the C++ Method.

**Parameters**

[Required if Parameters exist] A complete list of all parameters for the C++ Method.

**Remarks**

[Optional] If it has not already been covered under other entry headings, include here anything that would help the reader to a better understanding of the C++ Method's purpose and function.

**Limitations**

[Optional] Define any limitations that could be assigned to the C++ Method.

**Return Value**

**[Required]** List all possible return values of the C++ Method.

**Portability**

[Optional] If applicable, describe whether C++ Method is portable to other versions or revisions.

**Example**

**[Required]** An example will often explain more than any written statement how and why the item functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related. Be sure to reference to object classes.

## 7.3 Constant = <Constant Name>

**Overview**

**[Required]** A short description of the constant definition. This is used as the heading for locating the item in the On-Line help systems. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Syntax**

**[Required]** A complete listing of the constant.

Example:

```
 #define EBUS_WAIT_INFINITE     ((long)-1)
Example:
 Const int   MY_VAR   14;
```

**Defined In**

**[Required]** The name of the .H file where the constant is defined.

**Description**

**[Required]** This is a more detailed description of the constant's definition than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that.

**Remarks**

[Optional] If it has not already been covered under other entry headings, include here anything that would help the reader to a better understanding of the constant's purpose and function.

**Example**

[Optional] An example will often explain more than any written statement how and why the constant functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related.

## 7.4 Data Type = <Data Type Name>

**Overview**

**[Required]** A short description of the data type's functionality. This is used as the heading for locating the item in the On-Line help systems. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Syntax**

**[Required]** A complete listing of the data type. For example:

```
typedef struc
{
   char Name[] = “Enter your name here\0”;
   UINT8 Age;
   enum  Sex = {Male, Female};
} PersonnelRecord
```

**Defined In**

**[Required]** The name of the file where data type is defined.

**Description**

**[Required]** This is a more detailed description of the item's function than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that.

**Fields**

[Required for structures] List of fields, if the data type is a structure.

**Elements**

[Required for enumerations] List of elements ,if the data type is an enumeration.

**Portability**

[Optional] If applicable, describe whether data type is portable to other versions or revisions.

**Example**

[Optional] Highly recommended. An example will often explain how the data type is used more than any written statement.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related.

## 7.5 Function = <Function name>

**Overview**

**[Required]** A short description of the function's purpose. This is used as the heading for locating the item in the On-Line help systems. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Prototype**

**[Required]** A complete listing of the function. Example:

```
IOBusStatus IOEventCreate
(
  IMC_HDL        Hdl,
  char          *EventName,
  IOEventData   *EventData,
  IOHdl         *EventHdl
);
```

**Prototype In**

**[Required]** The name of the .H file that contains the prototype of the function.

**Description**

**[Required]** This is a more detailed description of the item's function than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first, then develop the shorter Overview statement from that.

**Preconditions**

[Optional] If applicable, one or more possible preconditions that are required by the item.

**Parameters**

[Optional] A complete list of all parameters for the item.

**Remarks**

[Optional] If it has not already been covered under other entry headings, include here anything that would help the reader to a better understanding of the item's purpose and function.

**Limitations**

[Optional] Define any limitations that could be assigned to the item.

**Return Value**

**[Required]** List all possible return values of the item.

**Example**

**[Required]** An example will often explain more than any written statement how and why the item functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related. Be sure to include all relevant data types listed in the function's syntax.

## 7.6 Function Type = <Function Type Name>

**Overview**

**[Required]** A short description of the function type's purpose. This is used as the heading for locating the item in the On-Line help systems. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Prototype**

**[Required]** A complete listing of the function type. Example:

```
typedef  SINT32 (* IOPointEventFilterFunc)
  (
    char     * IOPointName,
     void     * PointValue
  );
```

**Defined In**

**[Required]** The name of the .H file where function type is defined.

**Description**

**[Required]** This is a more detailed description of the item's function than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that. The description is used in the Reference Manual.

**Preconditions**

[Optional] If applicable, one or more possible preconditions that are required by the function type.

**Parameters**

[Optional] A complete list of all parameters for the function type.

**Remarks**

[Optional] If it has not already been covered under other entry headings, include here anything that would help the reader to a better understanding of the function types purpose.

**Limitations**

[Optional] Define any limitations that could be assigned to the function type.

**Return Value**

**[Required]** List all possible return values of the function type.

**Portability**

[Optional] If applicable, describe whether item is portable to other versions or revisions.

**Example**

**[Required]** An example will often explain more than any written statement how and why the item functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related. Be sure to include all relevant data types listed in the function type's syntax.

## 7.7 Macro = <Macro Name>

**Overview**

**[Required]** A short description of the macro's functionality. This is used as the heading for locating the item in the On-Line help systems. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Syntax**

**[Required]** A complete listing of the data type. Example:

```
TIMER_ACTION_POST_AND_COUNT32
(
  KerMsgQHdl queue,
  KerMsg msg,
  SINT32 *adr,
  SINT32 step
);
```

**Prototype In**

**[Required]** The name of the .H file that contains the prototype of the macro.

**Description**

**[Required]** This is a more detailed description of the macro's function than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that. The description is used in the Reference Manual.

**Parameters**

**[Required]** A complete list of all parameters for the macro.

**Remarks**

[Optional] If it has not already been covered under other entry headings, include here anything that would help the reader to a better understanding of the macro's purpose and function.

**Limitations**

[Optional] Define any limitations that could be assigned to the macro.

**Portability**

[Optional] If applicable, describe whether macro is portable to other versions or revisions.

**Example**

**[Required]** An example will often explain more than any written statement how and why the item functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related.

## 7.8 Module = <Module Name>

**Overview**

**[Required]** A short description what the module. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Description**

**[Required]** This is a more detailed description of the object than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related.

## 7.9 Object = <Object Name>

**Overview**

**[Required]** A short description of the object. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Description**

**[Required]** This is a more detailed description of the object than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that.

**Example**

[Optional] An example will often explain more than any written statement how and why the item functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related.

## 7.10 Term = <Term Name>

**Overview**

**[Required]** A short description of the term. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Description**

**[Required]** This is a more detailed description of the term than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that.

**Example**

[Optional] An example will often explain more than any written statement how and why the item functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related.

## 7.11 Variable = <Variable Name>

**Overview**

**[Required]** A short description of the variable. In order to make the On-Line help menu systems easy to navigate and simple to use, do not make your Overview entries longer than 65 characters in Windows NT-based software and 40 characters in DOS-based software.

**Syntax**

**[Required]** A complete listing of the variable type.

**Defined In**

**[Required]** The name of the .H file where function type is defined.

**Description**

**[Required]** This is a more detailed description of the variable than noted in the Overview. Since there are no limits on the length of the description, you may find it easier to write the description first then develop the shorter Overview statement from that.

**Example**

[Optional] An example will often explain more than any written statement how and why the item functions.

**See Also**

**[Required]** Use this section to refer the reader to any other items which are related.