|  |
| --- |
| Microsoft Surface |
| DMF Modules Overview |
| V1.1.1 |

|  |
| --- |
| Sam Tertzakian and Rajesh Gururaj  8-7-2018 |

**Document Change Log**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Description** |
| April 10, 2018 | 1.0.0 | Sam Tertzakian | Initial version. |
| June 4, 2018 | 1.1.0 | Sam Tertzakian | Update for changes due to DMF rename. |
| August 7, 2018 | 1.1.1 | Sam Tertzakian | Update for recent additions. |

**Document Conventions**

|  |  |
| --- | --- |
| **Convention** | **Meaning** |
| Child Module | A Module that is instantiated by another Module. |
| Client | The user of a Module; the code being served by a Module. It can either be a Parent Module or a Client Driver. |
| Client Driver | A WDF device driver that uses DMF. |
| DMF | Driver Module Framework |
| Library | A library of DMF Modules. |
| Module | A DMF Module. A set of code that has a specific structure that is compatible with the DMF framework. |
| Parent Module | A Module that instantiates a Child Module. |
| WDF | Windows Driver Framework (KMDF or UMDF). |
| WDM | Windows Driver Model. The layered driver model that the Windows operating system uses to organize device drivers. |

**Documentation Guide**

This document is part of a family of documents that explain DMF…

|  |  |
| --- | --- |
| **Document Title** | **Purpose** |
| *Device Driver Writing Considerations: An Introduction to DMF* | This document discusses issues device driver authors need to consider when writing device drivers. Furthermore, this document lays out the case for why DMF exists and why it can help device driver programmers. |
| *Driver Module Framework (DMF)* | This document explains the DMF API and how to use it. As you read this document, please have access to the source code so you can have a better view of how the APIs are used. |
| *BranchTrack* | This document explains what BranchTrack is and how to use it. (BranchTrack requires DMF.) As you read this document, please have access to the source code so you can have a better view of how the APIs are used. |
| *DMF Modules Overview* | This document lists all the DMF Modules provided by the Framework in the Modules.Library Library. A summary of each Module is provided. |

Contents

[1 DMF Modules List 6](#_Toc521402814)

[2.1 Data Structures 8](#_Toc521402815)

[2.1.1 Data Structures Module List 8](#_Toc521402816)

[2.2 Buffers 9](#_Toc521402817)

[2.2.1 Buffers Module List 9](#_Toc521402818)

[2.3 Task Execution 11](#_Toc521402819)

[2.3.1 Task Execution Module List 11](#_Toc521402820)

[2.4 Targets 12](#_Toc521402821)

[2.4.1 Targets Module List 12](#_Toc521402822)

[2.5 User Notification 14](#_Toc521402823)

[2.5.1 User Notification Module List 14](#_Toc521402824)

[2.6 Miscellaneous 15](#_Toc521402825)

[2.6.1 Miscellaneous Module List 15](#_Toc521402826)

[2.7 Features 17](#_Toc521402827)

[2.7.1 Features Module List 17](#_Toc521402828)

[2.8 Template 18](#_Toc521402829)

[2.8.1 Template Module List 18](#_Toc521402830)

# DMF Modules List

This section contains a list all the DMF Modules in the DMF Library sorted by name. The Module’s category membership is indicated.

|  |  |
| --- | --- |
| **Module Name** | **Module Category** |
| *DMF SelfTarget* | Targets |
| *Dmf\_AcpiNotification* | Miscellaneous |
| *Dmf\_AcpiTarget* | Targets |
| *DMF\_AlertableSleep* | Miscellaneous |
| *DMF\_BufferPool* | Buffers |
| *DMF\_BufferQueue* | Buffers |
| *Dmf\_ButtonTargetViaMsGpio* | Miscellaneous |
| *Dmf\_ContinuousRequestTarget* | Targets |
| *Dmf\_CrashDump* | Miscellaneous |
| *Dmf\_DeviceInterfaceTarget* | Targets |
| *Dmf\_GpioTarget* | Targets |
| *DMF\_HashTable* | Data Structures |
| *Dmf\_HidTarget* | Targets |
| *Dmf\_I2cTarget* | Targets |
| *DMF\_IoctlHandler* | Miscellaneous |
| *Dmf\_NotifyUserWithEvent* | User Notification |
| *Dmf\_NotifyUserWithRequest* | User Notification |
| *DMF\_Pdo* | Miscellaneous |
| *Dmf\_PingPongBuffer* | Buffers |
| *Dmf\_QueuedWorkitem* | Task Execution |
| *DMF\_RequestTarget* | Targets |
| *DMF\_Registry* | Miscellaneous |
| *Dmf\_ResourceHub* | Targets |
| *DMF\_RingBuffer* | Buffers |
| *DMF\_ScheduledTask* | Task Execution |
| *DMF\_SerialTarget* | Targets |
| *Dmf\_SmbiosWmi* | Miscellaneous |
| *Dmf\_SpiTarget* | Targets |
| *DMF\_Thread* | Task Execution |
| *Dmf\_ThreadedBufferQueue* | Buffers |
| *Dmf\_VirtualHidDeviceVhf* | Miscellaneous |
| *DMF\_VirtualHidKeyboard* | Miscellaneous |
| *Dmf\_Wmi* | Miscellaneous |

DMF Module Categories

This section provides tables that list the types of Modules provided by DMF as well as a summary of each Module’s purpose.

## Data Structures

This category contains Modules that provide support for commonly used data structures. Some Methods provide additional functionality for operations that are commonly done with these data structures.

### Data Structures Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *DMF\_HashTable* | Provides a classic hash table. The size of the table as well as the size of each record is defined by the Client. |

## Buffers

This category contains Modules that provide support for commonly used types of buffers.

### Buffers Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *DMF\_BufferPool* | Provides a list of zero or more pre-allocated buffers. The buffers each have an optional Client Context. The buffers also have sentinels to guard against buffer overrun. These sentinels are checked every time the buffer is used so that when an overrun error happens it can be immediately detected. Furthermore, each buffer can be added to the list in such a way that it is automatically removed from the list after a certain time (on a buffer by buffer basis). This data structure is used by many other Modules. The list is backed up by an optional lookaside list that can allocate buffers when the list of buffers becomes empty and new buffers are needed. |
| *DMF\_BufferQueue* | This data structure is composed of two DMF\_BufferPool instances. One is a list of pre-allocated empty buffers (Producer) and the other is an empty list (Consumer). This Module is often used when a Client needs to create a work queue. When the Client has work to add to the queue, the Client removes a pre-allocated buffer from the Producer, populates it with the work to be done and writes the buffer to the Consumer list. Later, the buffer is removed the from the Consumer list, the indicated work is completed and the buffer is returned to the Producer list. Of course, since this data structure is built using DMF\_BufferList, all operations are bounds checked to detect corruption. |
| *Dmf\_PingPongBuffer* | Provides a “ping-pong” buffer that allows the Client to easily parse unstructured data into packets where one buffer (ping) holds the last full buffer and the other buffer (pong) holds the current incomplete buffer. The Client can read from the complete buffer while another part of the Client code continues populating the incomplete buffer. |
| *DMF\_RingBuffer* | Provides a classic ring-buffer data structure. The number of ring-buffer entries as well as the size of each entry is provided by the Client. Additional enhancements include an “infinite” mode that automatically removes the oldest ring-buffer element when a new element is added to a full ring-buffer. Also, each record of the ring-buffer can be populated from a table of addresses and lengths. This is useful when reading/writing structures such as protocol headers where a single copy into the ring-buffer element’s buffer is not sufficient. Also, there is a Method that allows the Client to enumerate all the ring-buffer elements and, optionally, perform an operation on zero, one or more of the elements in the ring-buffer. |

|  |  |
| --- | --- |
| *Dmf\_ThreadedBufferQueue* | This Module uses DMF\_Thread and DMF\_BufferQueue to allow a Client to insert work into a work queue via multiple threads. When a Client calls the Method to add work to the work queue, a buffer is removed from the Producer list. Then, the buffer is populated with the work to be done and the buffer is added to the Consumer list. Then, the Thread’s work ready event is set. The corresponding thread callback function is called. That function retrieves the work to be done from the Consumer list and calls the Client callback function. That function does the work specified. Finally, the buffer is returned to the Producer list. Options are provided for caller’s threads to wait for the work to be completed or the work can be completed asynchronously. This is an extremely common device driver programming pattern. It ensures that all work is done in order of being requested, one operation at a time. |

## Task Execution

This category contains Modules that give the Client access to pre-written common device driver programming patterns related to (deferred) task execution.

### Task Execution Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *Dmf\_QueuedWorkitem* | This Module provides functionality similar to a WDFWORKITEM. However, it differs in these ways: 1. The deferred call will execute exactly the number of times that the Client has enqueued it. The Client does not need to track if the deferred call was already enqueued. The Module performs that housekeeping. 2. The order in which the deferred calls happen is exactly the in the order in which the calls are enqueued. 3. Client may wait for the deferred call to execute without explicitly creating and setting an event. |
| *DMF\_ScheduledTask* | This Module allows the Client to specify that an operation should happen a single time, either for the lifetime of the machine or for the current boot cycle. There are options for retrying the operation in case of failure or, even success. The Module takes care of all the housekeeping tasks needed to perform the required functionality. For example, the Module will remember that an operation has occurred on a machine by writing a flag to the Registry and automatically reading that same flag every time the Module is instantiated. |
| *DMF\_Thread* | This Module allows a Client to specify that a callback function is called whenever there is work for that callback function to perform. This Module creates a Thread and two events: 1. A stop event that tells the thread when it should stop. 2. A work event that indicates when work needs to be done. The thread waits on the two events. When the work event is set, the Client’s callback function is called so that the Client can do work. A Method is provided that tells the Module that work is ready to be done. This is a common programming pattern. |

## Targets

This category contains Modules that give the Client access to pre-written Modules that access various Hardware devices and buses via WDF IO Targets.

### Targets Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *DMF SelfTarget* | This Module allows a Client to send a WDFREQUEST to its own stack. This is important in some cases in order to cause a WDFREQEUST to be seen by filter drivers or other drivers in stack. |
| *Dmf\_AcpiTarget* | This Module allows a Client to query ACPI in various ways. This Module allows the Client to easily query/invoke/evaluate DSM methods. |
| *Dmf\_ContinuousRequestTarget* | This Module allows the Client to easily communicate with an underlying WDFIOTARGET. Two modes are supported: (1) The Client can either call synchronous routines using input/output buffers. (2) The Client can specify the number of asynchronous operations that are automatically created/sent/received to the underlying WDFIOTARGET. In the second case, the Client specifies the sizes of the input/output buffers. The Module creates the buffers as specified. Then, prior to sending an input buffer to the WDFIOTARGET, the Client’s input buffer callback function is called where the Client populates the input buffer that is sent. When an output buffer is received, it is sent to the Client’s output buffer callback function where the Client may retrieve the contents of the output buffer. |
| *Dmf\_DeviceInterfaceTarget* | The Module allows the Client to specify the GUID of a device interface The Module registers for a PnP Notification for when that device interface appears/disappears. When the device interface appears, the Module automatically opens the corresponding WDFIOTARGET. When the device interface disappears, the Module automatically closes the corresponding WDFIOTARGET. Methods are provided that allow the Client to send/receive WDFREQUESTS to that target. Furthermore, it is possible to set up a stream of asynchronous WDFREQUESTS that are automatically created/sent/received to the WDFIOTARGET. |
| *Dmf\_GpioTarget* | This Module allows the Client to communicate with the GPIO pins exposed by the GPIO WDFIOTARGET. This Module automatically looks for the GPIO resources and opens the appropriate targets based on settings set by the Client. |

|  |  |
| --- | --- |
| *Dmf\_HidTarget* | This Module allows the Client to communicate with an underlying HID WDFIOTARGET. Methods are provided that allow the Client to work with input/output/feature reports. This Module automatically opens HID WDFIOTARGETs as they appear. It queries each one to determine if it matches parameters specified by the Client. If so, the Module opens that WDFIOTARGET and prepares it for use by the Client. |
| *Dmf\_I2cTarget* | This Module allows the Client to communicate with an I2C bus that is exposed by SPB. The Client specifies the desired bus index. Methods are provided that allow the Client to read/write I2C data. |
| *Dmf\_RequestTarget* | This Module contains code that builds and send requests. It can be used with any WDFIOTARGET. |
| *Dmf\_ResourceHub* | This Module allows the Client to communicate with a Resource Hub WDFIOTARGET. The Module contains all the code need to parse the type resource and other information. |
| *DMF\_SerialTarget* | This Module allows the Client to specify the parameters of the Serial IO Stream target to open. Then, it provides Methods that expose the functionality of DMF\_RequestStream to that Serial IO Target. |
| *Dmf\_SpiTarget* | This Module allows the Client to communicate with an SPI bus that is exposed by SPB. The Client specifies the desired bus index. Methods are provided that allow the Client to read/write SPI data. |

## User Notification

This category contains Modules that give the Client the ability to notify User-mode processes of events.

### User Notification Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *Dmf\_NotifyUserWithEvent* | This Module allows the Client to easily set up an event in Kernel-mode that is also accessible in User-mode. It saves the Client from having to write code that is duplicated often. |
| *Dmf\_NotifyUserWithRequest* | This Module allows a Client to specify that is able to enqueue a specific number of WDFREQUESTs that are sent to the Client via an IOCTL. When the Client is ready to complete the WDFREQUESTs (because an event has occurred in the Client), a callback function is called for each pending WDFREQUEST where the Client can populate the output buffer and complete the WDFREQUEST. This programming pattern is useful when a Client wants to notify User-mode programs of events asynchronously. |

## Miscellaneous

This category contains Modules that give the Client support for miscellaneous device driver programming patterns.

### Miscellaneous Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *Dmf\_AcpiNotification* | This Module allows a Client to register for and receive asynchronous notifications via ACPI. The Client specifies the type of notification desired, the callback function the Module should call when the notification happens and the IRQL level at which the callback function is called. |
| *DMF\_AlertableSleep* | This Module allows a Client to cause a thread to sleep. However, the Client can cancel that sleep at any time. It is similar to a Sleep() function except that the Sleep() can be interrupted. |
| *Dmf\_ButtonTargetViaMsGpio* | This Module allows the Client to communicate with the device interfaces exposed by MSGPIOWIN32. These interfaces allow the Client to inject button and tablet related messages. |
| *Dmf\_CrashDump* | This Module allows the Client to easily write data to the Windows Crash Dump file when the system crashes. This Module exposes a ring-buffer so that the Client can simply write data to the ring-buffer at different checkpoints. (Older data is automatically removed when the ring-buffer becomes full.) If the system crashes the entire ring-buffer is reordered so that the earliest entry is listed first. Then, the entire ring-buffer is written to the crash dump file. Optionally, the Client can write other data to the Crash Dump file. |
| *DMF\_IoctlHandler* | This Module allows the Client to create a table of all the supported IOCTLs, including the minimum sizes of the input/output buffers. Each record can also specify a callback function as well as the required User-mode access rights. The Module retrieves the input/output buffers, validates them, validates access rights (optional) and then, if all those validations succeed, the corresponding callback function is called. The callback function is able to directly use the input/output buffers. In the case where the minimum buffer sizes are variable, the callback function is able to validate the length. Finally, the callback function can return STATUS\_PENDING to indicate that the WDFREQUEST is held by the Client. Otherwise, the Module automatically completes the WDFREQUEST after the callback function executes. |
| *DMF\_Pdo* | This Module allows the Client to easily create PDOs. |

|  |  |
| --- | --- |
| *DMF\_Registry* | This Module has many functions that allow the Client to work easily with the Registry. It has Methods that allow the Client to specify various conditions under which operations that depend on Registry entries will occur. It also allows the Client to easily query and validate Registry entries using a single line of code. Furthermore, it has functions that allow the Client to write an entire tree of Registry entries listed in a table. |
| *Dmf\_SmbiosWmi* | This Module allows the Client to read the SMBIOS data which the Module retrieves using WMI. |
| *Dmf\_VirtualHidDeviceVhf* | This Module is a wrapper around the VHF API. Modules use this Module as a Child Module to create virtual HID devices. Those parent Modules expose Methods that are specific to that virtual HID device. |
| *DMF\_VirtualHidKeyboard* | This Module is an example of how to create a Virtual HID device using Dmf\_VirtualHidDeviceVhf. |
| *Dmf\_Wmi* | This Module allows the Client work with the WMI API easily. |

## Features

This category contains Modules that DMF itself uses to provide functionality to Modules and Client drivers that use DMF. In this way, the Client does not need to specifically instantiate the Module as DMF automatically does so.

### Features Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *Dmf\_BranchTrack* | This Module provides an API that allows the Client to annotate branches of code in a driver. See “BranchTrack” for more information. |
| *Dmf\_Bridge* | This Module is used internally by DMF. Clients do not, and should not, instantiate this Module. |

## Template

This category contains Modules that programmers can use as templates for other Modules. Usually programmers find Module similar to the Module to be written and copy that. These templates are provided mostly to give easy access to signatures of functions and to show how a well known WDF driver looks like as a DMF driver.

### Template Module List

|  |  |
| --- | --- |
| **Module Name** | **Summary** |
| *Dmf\_Template* | This Module a Module that simply contains all the sections that a Module’s file contains. It also contains all the signatures for all the possible DMF/WDF callback functions. Programmers can copy these signatures when adding a new callback to a Module. |
| *Dmf\_ToasterBus* | This Module contains the functionality exposed by the WDF ToasterBus example. |