# MPQ Programmer API
# Application Interface User Guide
# RPM Systems Corporation

| | |
|---|---|
| December 15, 2010 | Initial Release |
| February 10, 2014 | Added OverlayData function |

*Overview* ....3
Status....3
Invoking the API....5
API Function Details....6
Open() ....6
SelectProgrammer() ....6
PortEnables()....7
Program() ....7
Status() ....8
StatusDecodes() ....11
Sample Application....11

# *Overview*

The MPQ API consists of a Windows .NET compliant (managed) class module, distributed as a dynamic- link library (DLL). The API provides the functionality necessary for a user to write native code to access the programmer, initiate and monitor programming operations, and report completion status. The API does not provide the functionality to load or manage image information in the programmer. The MPManager software is required for these operations.

It is possible to user the API to manage programming on multiple programmers on a given communication port, commonly referred to as an array. Arrays may be constructed using RPM's MPQ-AIB-USB array interface board. This product provides a single USB-mapped COM interface on the PC side, supporting multiple programmers on an RS-485 bus on the programmer side. Programmers in an array are distinguished from one another by a programmer address.

The API provides the following functions:

| | |
|---|---|
| Open() | Opens a communication port to the programmer(s) |
| SelectProgrammer() | Selects, by address, the programmer to which commands are being directed |
| Close() | Closes the communication port to the programmer(s) |
| PortEnables() | Selects which programmer ports will be enabled for Program() operations |
| Program() | Initiates a programming cycle on the selected programmer |
| Status() | Returns port status for the selected programmer |
| StatusDecode() | Decodes a numeric port status value into a status string |
| OverlayData() | Allows programming of sections of memory without a complete erase |

# Status

Each API call returns a *uint32* iSTS value. The possible values for iSTS are exported as constants from the MPQ API. The possible values for iSTS are provided below.

| | |
|---|---|
| STS_COMM_OPEN_FAIL | Communications port open failed. Typically this occurs if an invalid COM port selection is provided, or another application is using the selected COM port. |
| STS_SUCCESS | Command completed successfully |
| STS_COMM_RCV_FAIL | Returned if a low-level communication failure occurs while reading data from the COM port, for example, as a result of reading from a COM port which has not been opened. |
| STS_COMM_XMT_FAIL | Returned if a low-level communication failure occurs while writing data to the COM port, for example, as a result of writing to a COM port which has not been opened. |
| STS_COMM_RCV_TIMEOUT | Returned if a valid command response is not received from MPQ within a defined timeout period. The timeout period for each command is set by the API, and is typically 500mS. Longer values are used for certain API functions, such as changing modes. |
| STS_COMM_PROGRAMMER_NOT_SELECTED | The API requires that a programmer be selected, using the SelectProgrammer() function, before communication can be directed to it. This status is returned if an API is invoked before a programmer has been selected. |
| STS_CMD_RESP_ID | This is a protocol layer error indicating that a valid command response was received from the MPQ, but the response ID did not correspond to the command issued. |
| STS_CMD_RESP_LENGTH | This is a protocol layer error indicating that a valid command response was received from the programmer, but the length of the response data was not as expected. |

| | |
|---|---|
| STS_RESP_ADDRESS | This is a protocol layer error indicating that a valid command response was received but the device address did not correspond to the command issued. |
| RESP_MISSPARAM | This is an MPQ generated status which indicates that the command it received was missing one or more required parameters. |
| RESP_BADPARAM | This is an MPQ generated status which indicates that one or more command parameters were invalid. |
| RESP_BADCMD | This is an MPQ generated status which indicates that an unrecognized command was received. |
| RESP_BUSY | This is an MPQ generated status. It is issued to indicate that the received command cannot be executed because the MPQ is busy. This is typically returned if a Program() command is issued while the MPQ is already actively programming. |
| RESP_BADIMAGE | This is an MPQ generated status which indicates that the Program() command requested programming with an invalid Image #. MPQ supports four images, 1-4. |
| RESP_NOIMAGE | This is an MPQ generated status which indicates that the Program() command requested programming with a valid Image #, but the associated Image slot is either empty, or the image is corrupt. |

## Invoking the API

The MPQ_API class must be imported into the user application, and instantiated before it can be used.

```
Imports MPQ_API
```

```
Public Class UserApp
…
   Dim MyProgrammer As New MPQ_API.MPQ_API
…
End Class
```

In addition, files MPQ_API.DLL and MPQ_COMM.DLL must be included in the application path

# API Function Details

## Open()

The Open() function is used to open a communication port to the programmer(s). It is defined as follows:

```
Public Function Open(ByVal CommPort As String) As UInt32
```

CommPort is a string value indicating the communications port to be opened.

Example (vb.net):

```
ists = MyProgrammer.Open("COM1")        ' open COM port to programmer(s)
```

## SelectProgrammer()

The SelectProgrammer() function selects, by address, the programmer to which subsequent commands will be directed. It is defined as follows:

```
Public Function SelectProgrammer(ByVal AddressUB As Byte) As UInt32
```

AddressUB  is a byte value indicating the programmer address. MPQ programmers are typically shipped from the factory at address 1. For array applications, each programmer must have a unique address. The programmer address may be changed using the MPManager application.

Example (vb.net):

```
   ists = MyProgrammer.SelectProgrammer(1) 'select programmer address 1
```

## PortEnables()

MPQ provides four physical ports for connection to four targets in parallel. The PortEnables() function is used to select which of the ports will be enabled for subsequent programming operations. It is defined as follows:

```
Public Function PortEnables(ByVal bmEnablesUB As Byte) As UInt32
```

Parameter bmEnablesUB is a bit-mapped parameter which is encoded as follows:

| | | |
|---|---|---|
| Bit 0 | Port 1 Enable | 1 = Enabled, 0 = Disabled |
| Bit 1 | Port 2 Enable | 1 = Enabled, 0 = Disabled |
| Bit 2 | Port 3 Enable | 1 = Enabled, 0 = Disabled |
| Bit 3 | Port 4 Enable | 1 = Enabled, 0 = Disabled |

The API exports bit definitions for the four port enables which can be logically OR's to create bmEnablesUB, as well as a definition for ALL ports enabled.

| | |
|---|---|
| bmPORT1 | Include to enable Port 1 |
| bmPORT2 | Include to enable Port 2 |
| bmPORT3 | Include to enable Port 3 |
| bmPORT4 | Include to enable Port 4 |
| bmPORTALL | Include to enable all 4 ports |

Example (vb.net):

```
ists = MyProgrammer.PortEnables(MPQ_API.MPQ_API.bm_PORTALL) 'enable all ports
```

## Program()

MPQ provides four image slots in its internal flash, into which four unique images may be loaded using MPManager. The various parameters necessary for programming, such as the device type and device power supply voltage, are selected at the time the image is loaded. The MPQ API allows programming cycles using any of the four images to be initiated and monitored.

Programming is initiated using the Program() function. It is defined as follows:

```
Public Function Program(ByVal ImageUB As Byte) As UInt32
```

This function initiates programming of the image selected by ImageUB, on the ports enabled by the PortEnables() function. Valid values for ImageUB are between 1 and 4. Any other value of ImageUB will return a RESP_BADIMAGE status.

If the selected image is empty (no image has been loaded into the selected slot) or corrupt (fails a CRC check), Program() will return RESP_NOIMAGE.

Once programming has been successfully initiated, the Status() function is used to monitor programming progress.

Example (vb.net):

```
ists = MyProgrammer.Program(1)  'initiate programming with Image 1
```

## Status()

The Status() function returns the status of the four programmer ports. In addition, it returns one byte which indicates whether the programming operation is complete.

The format of this function is:

```
Public Function Status(ByRef PortStatusUB() As Byte)
```

PortStatusUB is a byte array at least 5 bytes in size, into which Status() writes the status informatione. The first byte (PortStausUB(0)) is a Busy status, defined as follows:

| | |
|---|---|
| Bit 0 | =1: Port 1 is Busy; =0: Port 1 is not busy |
| Bit 1 | =1: Port 2 is Busy; =0: Port 2 is not busy |
| Bit 2 | =1: Port 3 is Busy; =0: Port 3 is not busy |
| Bit 3 | =1: Port 4 is Busy; =0: Port 4 is not busy |

If PortStatusUB() returns a 0 value, the programming operation has completed on all ports.

Following the Busy status, one Port status byte is returned for each of the four ports (PortStatusUB(1) – PortStatusUB(4).. The possible port status values are exported by the API, and descriptions are provided in the following table. The table also indicates whether the particular status condition is a Busy or a Not Busy condition. Note that not all status values are valid for all types of programmers.

| | | |
|---|---|---|
| PSTS_DONE | Programming complete | Not Busy |
| PSTS_PGM | Programming in progress | Busy |
| PSTS_VFY | Verify in progress | Busy |
| PSTS_WFP | Waiting for Power. The programmer is waiting for power to be valid on this port. The programmer will wait 10 seconds for valid power before failing. | Busy |
| PSTS_VCC_OK | Valid power has been detected. This status is returned only in Power-on programming mode, when power has been detected on this port, but other enabled ports do not yet have valid power, so the programming cycle has not yet begun. | Busy |
| PSTS_DISABLED | The port is disabled | Not Busy |
| PSTS_READ | Reading target device | Busy |
| PSTS_CKSM | Performing device checksum | Busy |
| PSTS_EEPROM | Programming EEPROM | Busy |
| PSTS_EEPROM_VFY | Verifying EEPROM | Busy |
| PSTS_CFG | The programmer is reconfiguring its hardware for programming | Busy |
| PSTS_F_PWR | Failed power. Programmer failed to detect valid power on the port within 10 seconds of Program initiation. | Not Busy |
| PSTS_F_TO | Failed device timeout. Programmer timed out waiting for the device to return programming status (WAIT-AND-POLL). | Not Busy |
| PSTS_F_VFY | Failed Verify. Device data did not verify. | Not Busy |

| | | |
|---|---|---|
| PSTS_F_ID | Failed Device ID. Device ID read from the part did not match the target device. This could be an indication of an incorrect device, but it could also indicate a failure in the programmer serial data communication. | Not Busy |
| PSTS_F_VCC | Failed power. This failure is returned in Power-on programming mode if power is already detected on the device when Programming is initiated. | Not Busy |
| PSTS_F_OCD | OCD communication failed (Zilog Z8) | Not Busy |
| PSTS_F_RESET | Failed – Target reset is asserted. | |
| PSTS_F_COUNTEXP | Failed image count. The number of image impressions allowed for the selected image has been exhausted. This will only occur is image control is enabled when the image is loaded into the programmer. | Not Busy |
| PSTS_F_VFY_EEPROM | EEPROM Verify Failed | Not Busy |
| PSTS_F_FUSES | Fuse Verify Failed | Not Busy |
| PSTS_LOCKBITS | Failed programming Lock bits | Not Busy |
| PSTS_F_READPROT | Failed reading – read-protected device | Not Busy |
| PSTS_F_BADIMAGE | Failed – Image is bad. | Not Busy |

In addition, the API StatusDecode() function may be used to return a string representation of each valid port status.

Example (vb.net):

```
Dim PortStatusUB(4) As Byte
ists = MyProgrammer.Status(PortStatusUB)    'read ports status
```

## StatusDecodes()

The StatusDecode() function accepts a valid port-status code, as returned by the Status() function, and returns a string representation of the status value. This is provided as a convenience, to simplify UI implementations, and is not a required use. The function is defined as follows:

Public Function StatusDecode(ByVal StatusUB As Byte) As String

StatusUB is a standard port-status value. Note that the function does not return a standard iSTS value. If the port status value in StatusUB is not recognized, a value of "UNKNOWN" is returned.

## OverlayData()

The OverlayData() function allows a section of target memory to be programmed outside of the normal erase/program/verify flow. It is intended to allow target-device-specific data to be programmed into a device after completion of the normal gang-programming operation. NOTE that it is necessary that the area to be overwritten be unprogrammed (typically 0xFF) at the time the OverlayData command is issued. The function is defined as follows:

Public Function OverlayData (ByVal ImageUB As Byte, ByVal MemorySpaceUB As Byte, ByVal AddressUL As UInt32, ByVal LengthUB As Byte, ByVal DataUB() As Byte) As UInt32

| | |
|---|---|
| ImageUB | Image # to be used as a background pattern for the overlay |
| MemorySpaceUB | Memory space to be overlaid. Current only SPACE_EEPROM is supported. |
| AddressUL | Starting address for overlay |
| LengthUB | Number of bytes to be overlaid (64 max) |
| DataUB() | Byte array containing LengthUB bytes of data to be overlaid |

# Sample Application

A simple sample application, written in vb.net, is provided with the API release. The MPQ API is .NET compliant managed code and, as such, it can be invoked by any .NET application (C#, VCC, etc).