Speedway® Reader Application Series

# Speedway GPIO Applications

## Speedway GPIO

Impinj's Speedway® reader incorporates a general purpose input/output port that enables interaction with its external environment. Through this port the reader can be controlled by external stimuli and can trigger external events. External sensors connected to a GPIO input can generate a trigger that will activate the reader to identify tagged items, for example, when they are passing through a specific read zone. The GPIO outputs can drive signal lights to indicate a successful read. This application note will discuss the interface in some detail, will include example applications for both input and output circuitry, and will show some commercially available GPIO hardware.

## Speedway Reader I/O Ports and Status

Refer to Figure 1 for the location of the Speedway reader's GPIO port. This port consists of a female DB-25 connector. The GPIO connector contains an RS-232 serial interface, four 3.3/5V logic inputs, and eight 3.3V logic outputs. See Table 1 for the pin-out, Table 2 for the GPIO electrical specifications, and Figure 2 for the physical pin view.

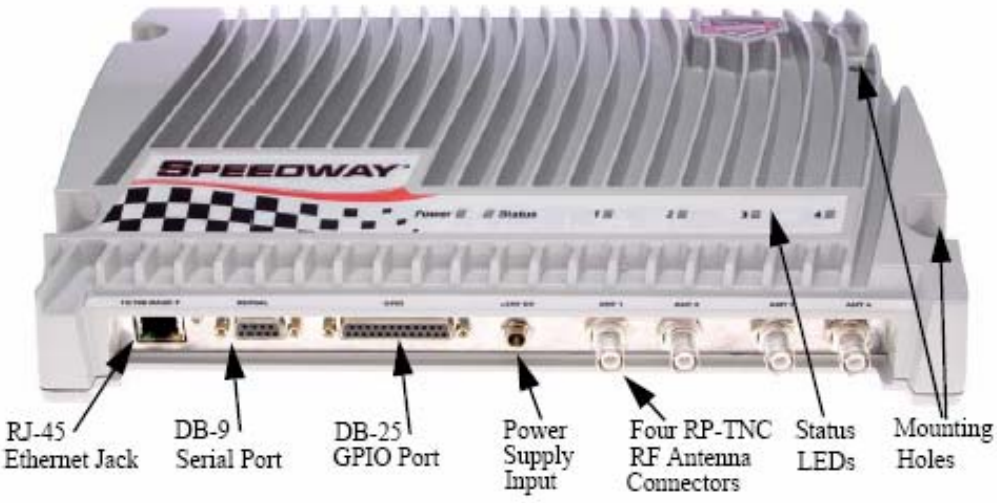



**Figure 1 Speedway GPIO Port Location**



Copyright © 2008, Impinj, Inc.
Impinj, Octane, and Speedway are either registered trademarks or trademarks of Impinj, Inc.

For more information, contact: rfid_info@impinj.com

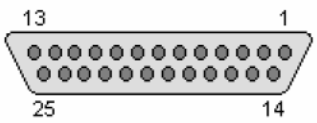
**Table 1 GPIO Pinout**

| Pin | I/O | Pin | I/O | Pin | I/O |
|---|---|---|---|---|---|
| 1 | No Connect | 10 | GPIN3 | 19 | GPOUT5 |
| 2 | RS-232 RXD | 11 | GPIN2 | 20 | No Connect |
| 3 | RS-232 TXD | 12 | GPIN1 | 21 | GPOUT6 |
| 4 | RS-232 CTS | 13 | GPIN0 | 22 | No Connect |
| 5 | RS-232 RTS | 14 | GPOUT0 | 23 | GPOUT7 |
| 6 | No Connect | 15 | GPOUT1 | 24 | No Connect |
| 7 | Signal Ground | 16 | GPOUT2 | 25 | No Connect |
| 8 | No Connect | 17 | GPOUT3 | | |
| 8 | No Connect | 18 | GPOUT4 | | |

**Caution:** Pins listed in Table 1 as "No Connect" must be left unconnected

**Table 2 GPIO Electrical Specifications**

| Pin | Parameter | Description | Min | Max | Unit | Conditions |
|---|---|---|---|---|---|---|
| GPIN[3:0] | $V_{IH}$ | HIGH-level input voltage | 2 | 5 | V | |
| GPIN[3:0] | $V_{IL}$ | LOW-level input voltage | 0 | 0.8 | V | |
| GPIN[3:0] | $I_{LI}$ | Input Leakage Current | -5 | 5 | µA | $V_{in}$ = 0–5V |
| GPIN[3:0] | $V_I$ | Input Voltage Range | -5 | 5 | V | No damage |
| GPOUT[7:0] | $V_{OH}$ | HIGH-level output voltage | 3 | 3.3 | V | $I_{OUT}$ = 100 µA |
| GPOUT[7:0] | $V_{OH}$ | HIGH-level output voltage | 2 | - | V | $I_{OUT}$ = 10 mA |
| GPOUT[7:0] | $V_{OL}$ | LOW-level output voltage | 0 | 0.25 | V | $I_{OUT}$ = -100 µA |
| GPOUT[7:0] | $V_I$ | Input voltage range | -5 | 5 | V | No damage |





Copyright © 2008, Impinj, Inc.

**Figure 2 GPIO Connector Physical View**

# GPIO Simplified Schematics

A simplified schematic for a single GPIO output is shown in Figure 3. Each of the eight output lines are derived from a 74LVC237 octal latch within the reader. These lines are connected to the GPIO connector through 100 Ohm resistors. Each output pin is bypassed to ground with a .01 µF capacitor.

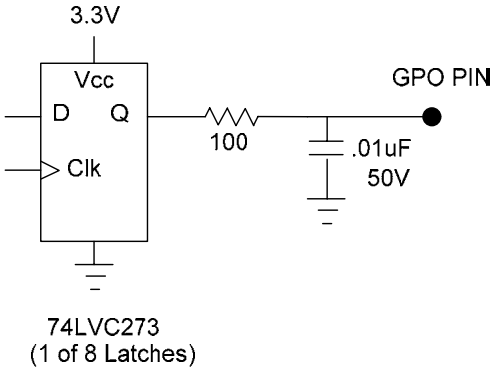



**Figure 3 Simplified Single Output Schematic**

Like the eight output pins, the four GPIO input pins are each bypassed to ground through a .01 µF capacitor, and then connected through a 100 Ohm resistor to the input ports of a 74LVC244 octal buffer. The simplified schematic is shown in Figure 4.

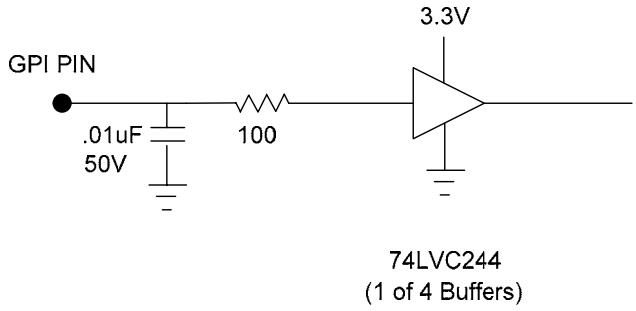



**Figure 4 Simplified Single Input Schematic**

The RS-232 serial interface schematic is shown in Figure 5. The outputs are controlled by user software loaded into the reader's microprocessor and can be used to provide status information.

Copyright © 2008, Impinj, Inc.

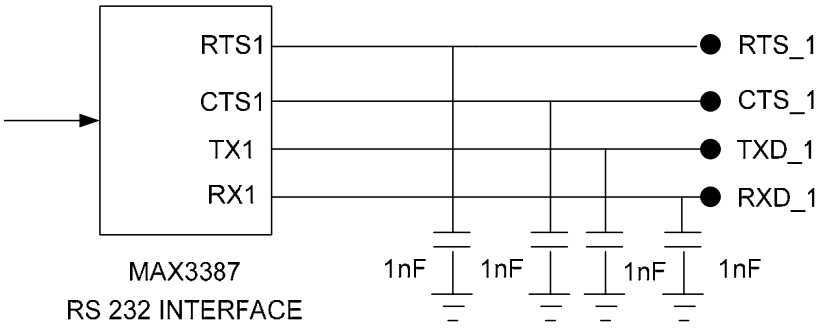



**Figure 5 Simplified RS-232 Port**

# Stack Light Applications Using GPIO Outputs

Stack light indicators are often used to indicate the operational status of equipment. The application circuit in Figure 6 shows how the reader GPIO outputs can be used to drive a stack light indicator. Outputs GPO0, GPO1, and GPO2 are connected through opto-isolators to LED drivers, which are used to switch the LED indicator lights. Opto-isolators protect the reader from voltage surges and ground currents, and are recommended in cases where external cables connect to the reader. Choosing an opto-isolator that can be driven directly from the GPO will simplify the interface. The OCP-PCT-114/E (see Appendix) was chosen for this reason. It requires a drive current between 10mA and 20mA at a corresponding input voltage of 1.2V. The internal resistor in series with the GPIO output provides current limiting, so the opto-isolator can be connected directly to the output.

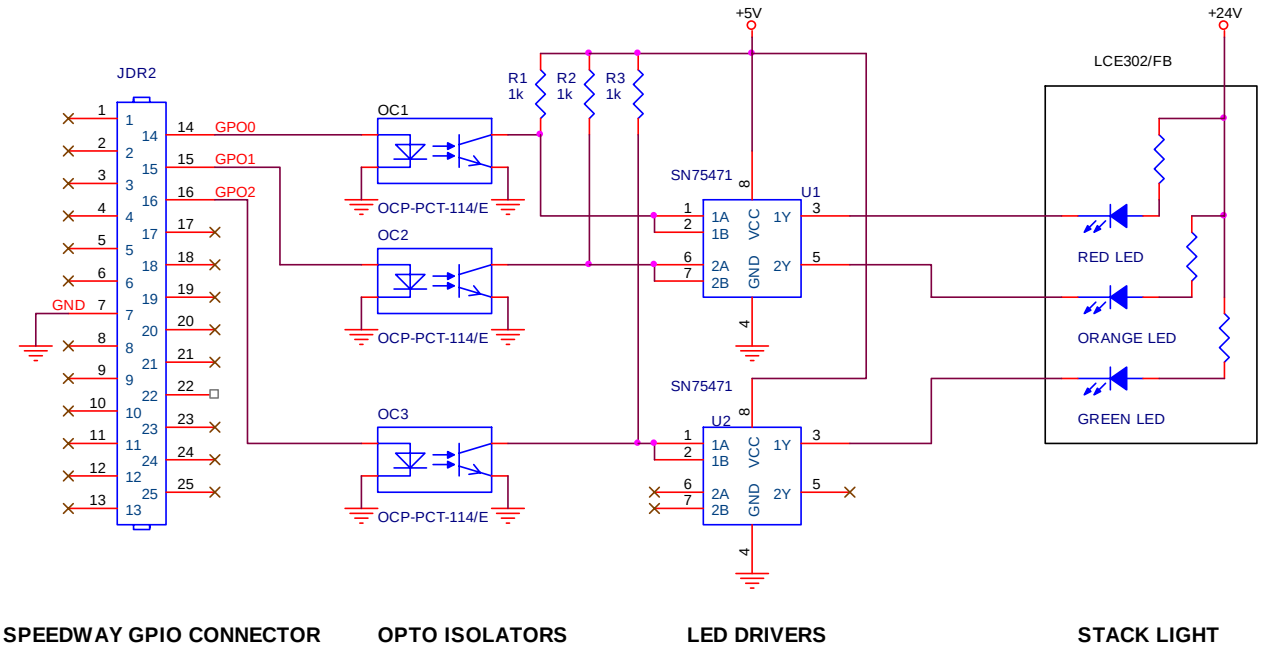



**Figure 6 Reader Stack Light Interface Schematic**



Copyright © 2008, Impinj, Inc.

# Demo Circuit to Test GPIO Inputs and Outputs

Figure 7 shows a simple demo circuit that will provide the user with a means to test the reader GPIO functions. Three LED lights are connected to GPO 0, GPO 1, and GPO 2. They can each be lit independently under reader program control by setting the appropriate pins high. GPO 0 is also connected to two pull-up resistors, so that when it is enabled, these resistors will provide a positive voltage to the two toggle switches. These switches are connected to GPI 0 and GPI 1; toggling the switches will cause the GPI inputs to switch between logic 1 and logic 0. The capacitors connected to the switch outputs provide a simple de-bounce function. These capacitors are completely discharged when the switch closes the first time. The time constant is selected such that during any successive contact bounce there is insufficient time for the voltage to rise high enough to trigger the input.

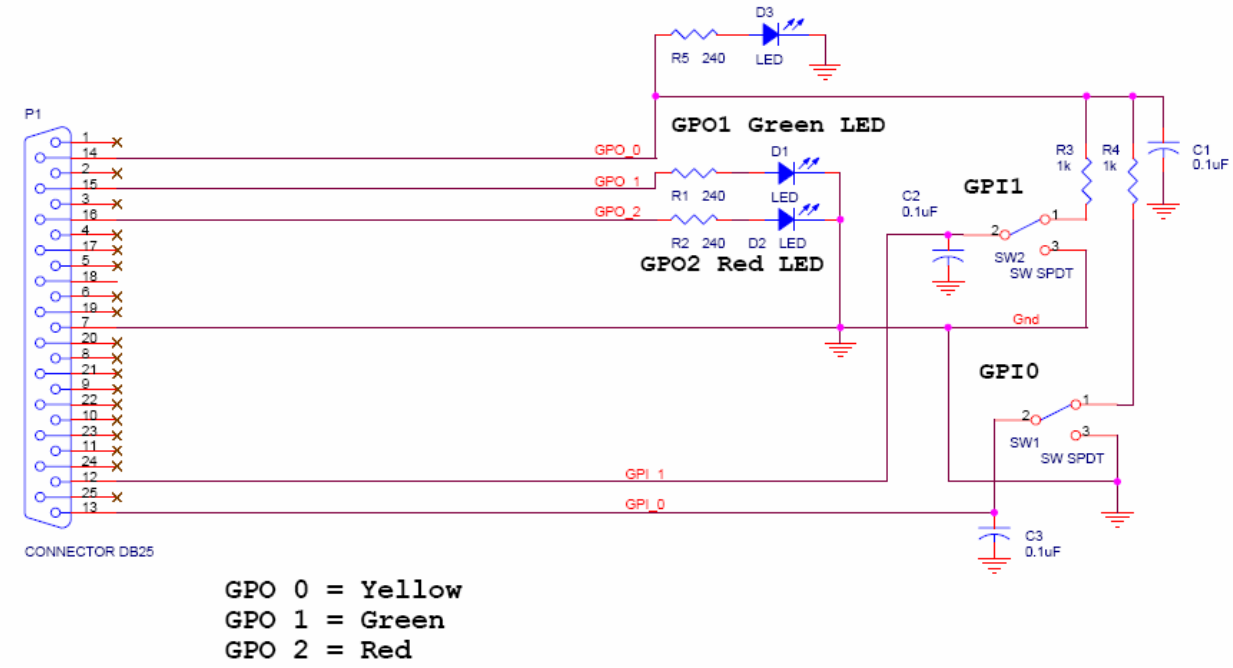



**Figure 7 GPIO Demo Circuit**

Copyright © 2008, Impinj, Inc.

# Compatible GPIO Interface Hardware

The following two products are examples of hardware that is compatible with the Speedway reader.

## Sensormatic® RFID GPIO[1]

The Sensormatic RFID GPIO Hardware Interface Board is compatible with the Speedway GPIO. A summary of its technical specifications follows. See www.sensormatic.com for more detailed information.

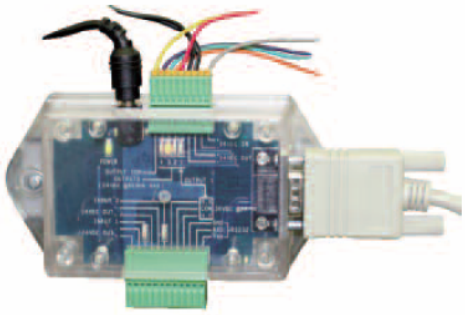
### Technical Specifications

**GPIO Inputs**
Number of discrete inputs: 2 or 4
Input Voltage Active (high): 6-30Vdc
Input Voltage Not Active (low): 0-3VDC

**GPIO Outputs**
Number of discrete outputs: 4 or 8
Output current (each): 300mA max.*
Output Voltage: 24Vdc
Relay switch load: 2A max @ 30VDC

**Power Supply Requirements**
24Vdc Certified Limited Power Source, Class 2
Able to share or use reader power supply
(*Total current supplied to devices connected to HIB must not exceed current rating of power supply)

**Physical/Environmental Specifications**
Dimensions: 13.3 cm x 9.5 cm x 6.9 cm
(5.25 in x 3.75 in x 2.75 in)
Weight: .18 kg (0.4 lb)
Operating Temperature Range: -10 to 50 deg C



Sensormatic® RFID GPIO
Hardware Interface Board

### Product Code

IDEQPCV013-1: GPIO HIB, Speedway (2 in/4 out)
IDEQPCV013-2: GPIO HIB, Speedway (4 in/8 out)

**Included Items**

- GPIO Hardware Interface Board (pictured above), includes two for IDEQPCV013-2
- 24Vdc Power Extension Cable .91 m (3 ft)

[1] Sensormatic is a registered trademark of Sensormatic Corporation. Technical specifications and the photograph are Copyright © 2006 Sensormatic Corporation. Used with permission.


Copyright © 2008, Impinj, Inc.

# Tacit Solutions® General Purpose I/O Box[2]

The Tacit Solutions General Purpose I/O Box is compatible with the Speedway GPIO. A summary of its technical specifications follows. See www.tacitsolutions.com for more detailed information.

## Specifications

- Requires +12v unregulated power supply
- 6 Inputs 0-30v DC
- 6 Outputs 0-250v @ 5A AC/DC mechanical relays normally open/normally closed, common (100K duty cycle). Max power 2.20 W @ 25° C.
- Temp -40° C to +80° C

[2] Tacit Solutions is a registered trademark of Tacit Solutions Corporation. Technical specifications and the photograph are Copyright © 2008 Tacit Solutions Corporation. Used with permission.

Copyright © 2008, Impinj, Inc.

# GPIO Programming

## Mach1 API

The Mach1 API provides an interface to the GPIO capabilities of the Speedway reader. In addition to commands for reading input and setting output states, it is possible to configure the Speedway reader to 'push' events to application software upon changes to the input states, reducing the need for continuous and inefficient input-polling schemes.

## Summary of Mach1 GPIO API Messages

For a full description of the GPIO capabilities in Mach1, refer to the *Mach1 Protocol Management Command Set* (see http://developer.impinj.com for documentation). A sample application is included in the Appendix, which demonstrates a simulated GPIO-triggered dock-door inventory.

### SetGPOCmd

Sets the state of one or more outputs.

### GetGPICmd

Reads the current state and configuration for all inputs.

### SetGPICmd

Configures the notification behaviour of one or more inputs.

### GPIAlertNtf

A notification sent from the reader indicating a GPI State has changed. These notifications must be enabled using the **SetGPICmd** command.

### LLRP

The EPCglobal Low Level Reader Protocol (LLRP) standard provides discrete control over GPIO ports in addition to higher level functionality, such as automatic GPIO-triggered inventory. With the Octane 3.0 firmware, the Speedway reader supports the full LLRP standard specification, including GPIO capabilities.

A complete software example demonstrating GPIO capabilities for LLRP is beyond the scope of this document. For more information please refer to the LLRP documentation and sample code at http://www.llrp.org.


Copyright © 2008, Impinj, Inc.

# Appendix

## Major Components Used in the Demonstration Circuits

| Description | Manufacturer | Manufacturer PN | Source | Source PN |
|---|---|---|---|---|
| Opto-Isolator | Lumex www.lumex.com | OCP-PCT114/E | DigiKey www.digikey.com | 67-1560-5-ND |
| Driver | Texas Instruments www.ti.com | SN75471P | DigiKey www.digikey.com | 296-9919-5-ND |
| Stack Light | Patlite www.patlite.com | LCE-302FB-RYG | Newark www.newark.com | 07H2201 |

## Mach1 GPIO Sample Code

The following code example demonstrates a simulated, GPIO-triggered, dock-door inventory. The application is designed to compile under Microsoft Visual C++, but could be ported to a different platform with relative ease.

### Listing 1: Mach1_GPIO_Dock_Door.cpp

```
/*
 ******************************************************************************
 *                                                                            *
 *                                                                            *
 *                                                                            *
 *                                                                            *
 * (c) Copyright Impinj, Inc. 2005 - 2007. All rights reserved.               *
 *                                                                            *
 ******************************************************************************
 *                                                                            *
 * This code is compatible with Mach1 2.8.0                                   *
 *                                                                            *
 ******************************************************************************
 */

#include "stdafx.h"
#include "Mach1_GPIO_Dock_Door.h"


////////////////////////////////////////////////////////////////////////////////
// GPIO Triggered Dock Door scenario
//
// Demonstrates:
// - Connecting to and configuring Speedway Reader
// - GPIO triggered Inventory, Reads unique EPCs for the specified time
```

Copyright © 2008, Impinj, Inc.

```
// - Selects the appropriate antenna based on the GPIO triggered
//
////////////////////////////////////////////////////////////////////////////////
int main(int argc, _TCHAR* argv[])
{

  ////////////////////////////
  //Connect to Speedway Reader
  ////////////////////////////
  //Specify Reader Host name or IP address here
  char *readerHost = "speedway";

  _reader = ConnectReader(readerHost);
  if (! _reader){
          printf("Failed to connect Speedway Reader for host '%s'\n",readerHost);
          exit(1);
  }

  ////////////////////////////
  // Start the Data thread
  ////////////////////////////
  _hDataThread = AfxBeginThread(DataThread, NULL , THREAD_PRIORITY_NORMAL);
  if (! _hDataThread){
          printf("Could not create data thread\n");
          exit(1);
  }

  ////////////////////////////
  //Reboot Reader's Modem
  ////////////////////////////
  BootModem();
  if (_lastMessageResult.resultCode !=0){
          printf("Failed to boot modem- result code %d",_lastMessageResult.resultCode);
          exit(1);
  }

  ////////////////////////////
  //Set Regulatory Region
  ////////////////////////////
  SetRegulatoryRegion( 0 );
  if ( _lastMessageResult.resultCode !=0 ){
          printf( "Failed to set regulatory region- result code %d", _lastMessageResult.resultCode
);
          exit( 1 );
  }
```


Copyright © 2008, Impinj, Inc.

```
  ////////////////////////////
  //Configure our GPIO settings
  ////////////////////////////

  //GPO 0 is used as a pull-up voltage for our GPIO test dongle
  ActivateGPO(0,1);
  //Reset GPI 0 and 1 Notifications to off
  ConfigureGPINotifications(0,0);
  ConfigureGPINotifications(1,0);
  //GPI 0 configure for low->high transitions
  ConfigureGPINotifications(0,1);
  //GPI 1 configure for low->high transitions
  ConfigureGPINotifications(1,1);

  //Set globals
  _inventoryIsRunning = false;
  _inventoryDurationMillis = 2000;

  while(1){
          printf("\nWaiting for GPI Trigger\n");
          WaitForSingleObject(_tagsInventoriedNotification,INFINITE);

          printf("\nInventoried tags\n");
          std::set<EPC>::iterator epcIter = _inventoriedTags.begin();
          //output the tags. tags are stored in an stl::set (_inventoriedTags)

          while (epcIter != _inventoriedTags.end()){
                  char EPCString[30];
                  FormatEPC(EPCString, epcIter->epcData, epcIter->epcLength);
                  printf("Tag: %s\n", EPCString);
                  ++epcIter;
          }
  }
  return 0;
}


READER *ConnectReader(char* readerHost)
{
  struct MAPI_CONNECT_ERROR connectResult;
  //Connect to Speedway Reader
  READER *reader = MAPI_Connect(readerHost,NULL,&connectResult);

  if (! reader){
          printf("Failed to connect Speedway Reader '%s' : ",readerHost);
          switch (connectResult.ErrorType){
```

Copyright © 2008, Impinj, Inc.

```
                case MAPI_CONNECT_ERROR_ADDRESS:
                        printf("The address is not valid.\n");
                        break;
                case MAPI_CONNECT_ERROR_CONNECTION:
                        printf("A connection could not be established.\n");
                        break;
                case MAPI_CONNECT_ERROR_BUSY_UNKNOWN:
                        printf("The Reader is currently busy with another unknown client.\n");
                        break;
                case MAPI_CONNECT_ERROR_BUSY_KNOWN:
                        printf("The Reader is currently busy with another known client.\n");
                        break;
                default:
                        printf("Error code %d received.\n", connectResult.ErrorType);
        }
  }
  return reader;
}


void BootModem()
{
  MAPI_FlushData( _reader );

  printf("Booting Modem...");
  //Issue Modem Reboot command
  struct MAPI_DATA response = MAPI_BootModemCmd(_reader);
  MAPI_Free(response.Data);

  bool bootComplete = false;
  while (! bootComplete){
        //WAIT for the boot result notification, signaled from thread
        WaitForSingleObject(_rebootNotification, INFINITE);

                switch (_lastMessageResult.resultCode){
                        case 0:
                                        printf("Modem reboot successful\n");
                                        bootComplete = true;
                                        break;
                        case 1: //Boot in progress
                                        printf("."); //print out a dot to show boot progress
                                        break;
                        case 2:
                                        printf("Modem reboot failed: Invalid Firmware\n");
                                        break;
                        case 3:
```


Copyright © 2008, Impinj, Inc.

```
                                        printf("Modem reboot failed: Hardware communication
timeout\n");
                                        break;
                        case 4:
                                        printf("Modem reboot failed: Boot failed due to unknown
hardware\n");
                                        break;
                        default:
                                printf("Modem reboot failed: unspecified error code
%d",_lastMessageResult.resultCode);
                }
  }
}

//Sets the regulatoryRegion
//Sets Reader Regulatory Region
void SetRegulatoryRegion( unsigned int regulatoryRegion )
{
          printf("Setting Regulatory Region %d...", regulatoryRegion);

          MAPI_FlushData(_reader);

          //Create the command structure and clear the memory
          struct MAPI_DATA_SET_REGULATORY_REGION_CMD command;
          memset(&command, 0, sizeof(command));

          //Set the required parameter
          command.RegulatoryRegion = regulatoryRegion;

          //Call the Reader and get the Response
          MAPI_DATA response = MAPI_SetRegulatoryRegionCmd(_reader, &command);

          //Check for an Invalid Command Notification
          switch (response.Type){
                  case MAPI_DATA_TYPE_INVALID_COMMAND_NTF:
                          {
                                  MAPI_DATA_INVALID_COMMAND_NTF *invalidCommand =
(MAPI_DATA_INVALID_COMMAND_NTF *)response.Data;
                                  printf("Received Invalid Command Notification while setting
regulatory Region %d\n", regulatoryRegion);
                                  printf("Reason Code = %d, Reader State = %d\n",invalidCommand-
>ReasonCode, invalidCommand->State);
                                  _lastMessageResult.messageType =
MAPI_DATA_TYPE_INVALID_COMMAND_NTF;
                                  _lastMessageResult.resultCode = invalidCommand->ReasonCode;
                                  MAPI_Free(invalidCommand);
                                  break;
                          }
                  case MAPI_DATA_TYPE_SET_REGULATORY_REGION_RSP:
```

Copyright © 2008, Impinj, Inc.

```
                        {
                                MAPI_DATA_SET_REGULATORY_REGION_RSP *setRegulatoryRegionRsp;
                                setRegulatoryRegionRsp = (MAPI_DATA_SET_REGULATORY_REGION_RSP
*)response.Data;
                                _lastMessageResult.messageType =
MAPI_DATA_TYPE_SET_REGULATORY_REGION_RSP;
                                _lastMessageResult.resultCode = setRegulatoryRegionRsp->ResultCode;
                                MAPI_Free(setRegulatoryRegionRsp);
                                if (_lastMessageResult.resultCode != 0){
                                        printf("Regulatory Region %d not supported. Result Code:
%d\n",regulatoryRegion, _lastMessageResult.resultCode);
                                }
                                break;
                        }
        }
        //WAIT for the asynchronous notification
        WaitForSingleObject(_regulatoryRegionChangeNotification,INFINITE);

        //result stored in global object
        switch(_lastMessageResult.resultCode){
                case 0:
                {
                        printf("Regulatory Region Set Successfully\n");
                        break;
                }
                case 1:
                {
                        printf("Regulatory Region not set- no valid regulatory calibration\n");
                        break;
                }
                case 2:
                {
                        printf("Regulatory Region not set- error setting regulatory region\n");
                        break;
                }
                default:
                        printf("Regulatory Region not set- unspecified error
%d\n",_lastMessageResult.resultCode);
        }
}


//Sends the Start Inventory Command
void StartInventory()
{
  struct MAPI_DATA_INVENTORY_CMD command;
  memset(&command, 0, sizeof(command));
```



Copyright © 2008, Impinj, Inc.

```
  MAPI_FlushData(_reader);
  MAPI_DATA response = MAPI_InventoryCmd(_reader, &command);

  switch (response.Type){
         case MAPI_DATA_TYPE_INVALID_COMMAND_NTF:
                {
                       MAPI_DATA_INVALID_COMMAND_NTF *invalidCommand =
(MAPI_DATA_INVALID_COMMAND_NTF *)response.Data;
                       _lastMessageResult.messageType = MAPI_DATA_TYPE_INVALID_COMMAND_NTF;
                       _lastMessageResult.resultCode = invalidCommand->ReasonCode;
                       MAPI_Free(invalidCommand);
                       break;
                }
         case MAPI_DATA_TYPE_INVENTORY_RSP:
                {
                       MAPI_DATA_INVENTORY_RSP *inventoryResponse = (MAPI_DATA_INVENTORY_RSP
*)response.Data;
                       _lastMessageResult.messageType = MAPI_DATA_TYPE_INVENTORY_RSP;
                       _lastMessageResult.resultCode = inventoryResponse->ResultCode;
                       MAPI_Free(inventoryResponse);
                }
  }
}

void StopInventory()
{
  MAPI_DATA response = MAPI_ModemStopCmd(_reader);

  switch(response.Type){
         case MAPI_DATA_TYPE_INVALID_COMMAND_NTF:
         {
                MAPI_DATA_INVALID_COMMAND_NTF *invalidCommand = (MAPI_DATA_INVALID_COMMAND_NTF
*)response.Data;
                _lastMessageResult.messageType = MAPI_DATA_TYPE_INVALID_COMMAND_NTF;
                _lastMessageResult.resultCode = invalidCommand->ReasonCode;
                MAPI_Free(invalidCommand);
                break;
         }
         case MAPI_DATA_TYPE_MODEM_STOP_RSP:
         {
                MAPI_DATA_MODEM_STOP_RSP *modemStopRsp = (MAPI_DATA_MODEM_STOP_RSP
*)response.Data;
                MAPI_Free(modemStopRsp);
                _lastMessageResult.messageType = MAPI_DATA_TYPE_MODEM_STOP_RSP;
                _lastMessageResult.resultCode = 0;
                break;
```

Copyright © 2008, Impinj, Inc.

```
        }
  }
  //Wait for the modem to stop
  WaitForSingleObject(_modemStoppedNotification, INFINITE);
}

void ConfigureGPINotifications(unsigned char gpi, unsigned char config){

  MAPI_DATA_SET_GPI_CMD command;
  memset (&command,0,sizeof(command));

  //configure for both HIGH->LOW or LOW->HIGH transitions
  switch(gpi){
        case 0:
              MAPI_SetOption(&command, MAPI_DATA_SET_GPI_CMD_OPT_GPI0_CONFIG);
              command.Gpi0Config = config;
              break;

        case 1:
              MAPI_SetOption(&command, MAPI_DATA_SET_GPI_CMD_OPT_GPI1_CONFIG);
              command.Gpi1Config = config;
              break;

        case 2:
              MAPI_SetOption(&command, MAPI_DATA_SET_GPI_CMD_OPT_GPI2_CONFIG);
              command.Gpi2Config = config;
              break;

        case 3:
              MAPI_SetOption(&command, MAPI_DATA_SET_GPI_CMD_OPT_GPI3_CONFIG);
              command.Gpi3Config = config;
              break;

        default:
              printf("Cannot set GPI configuration: Invalid GPI specified %d", gpi);
              exit(1);
  }
  struct MAPI_DATA_SET_GPI_RSP *response;


  response = (struct MAPI_DATA_SET_GPI_RSP *)(MAPI_SetGpiCmd(_reader,&command).Data);

  unsigned char resultCode = response->ResultCode;
  MAPI_Free(response);

  switch (resultCode){
```


Copyright © 2008, Impinj, Inc.

```
        case 0:
                break;
        case 1:
                printf("Could not configure GPI: The GPI specified (%d) is not supported on this
hardware",gpi);
                break;
        case 2:
                printf("Could not configure GPI: The GPI specified (%d) is currenty active",gpi);
                break;
        default:
                printf("Could not configure GPI %d: Unspecified resultCode received: %d", gpi,
resultCode);
  }
  if (0 != resultCode) exit(1);
}

void ActivateGPO(unsigned char gpo, unsigned char state){

  MAPI_DATA_SET_GPO_CMD command;
  memset(&command,0,sizeof(command));
  switch (gpo){
        case 0:
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO0_CONFIG);
                command.Gpo0Config = (state ? 1 : 0);
                break;
        case 1:
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO1_CONFIG);
                command.Gpo1Config = (state ? 1 : 0);
                break;
        case 2:
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO2_CONFIG);
                command.Gpo2Config = (state ? 1 : 0);
                break;
        case 3:
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO3_CONFIG);
                command.Gpo3Config = (state ? 1 : 0);
                break;
        case 4:
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO4_CONFIG);
                command.Gpo4Config = (state ? 1 : 0);
                break;
        case 5:
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO5_CONFIG);
                command.Gpo5Config = (state ? 1 : 0);
                break;
        case 6:
```

Copyright © 2008, Impinj, Inc.

```
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO6_CONFIG);
                command.Gpo6Config = (state ? 1 : 0);
                break;
        case 7:
                MAPI_SetOption(&command, MAPI_DATA_SET_GPO_CMD_OPT_GPO7_CONFIG);
                command.Gpo7Config = (state ? 1 : 0);
                break;
        default:
                printf("Invalid GPO requested %d",gpo);
                exit(1);
  }


  struct MAPI_DATA_SET_GPO_RSP *response;
  response = (struct MAPI_DATA_SET_GPO_RSP *)(MAPI_SetGpoCmd(_reader, &command).Data);
  unsigned char resultCode = response->ResultCode;

  MAPI_Free(response);

  if (0 != resultCode){
        printf("Failed to set GPO %d with value %d", gpo,state);
        exit(1);
  }
}


//Inventory timer thread
UINT InventoryTimerThread(LPVOID Arg)
{
  _inventoryIsRunning = true;
  //erase our set of inventoried tags
  ResetEvent(_tagsInventoriedNotification);
  _inventoriedTags.clear();


    unsigned char enabledAntennas = 0;
  if (0 == _lastInputTriggered){
        enabledAntennas = 3;  // binary 0011: set antennas 1 and 2
  }
  if (1 == _lastInputTriggered){
        enabledAntennas = 12; // binary 1100: set antennas 3 and 4
  }

  MAPI_DATA_SET_ANTENNA_CMD setAntennaCmd;
  memset(&setAntennaCmd,0,sizeof(setAntennaCmd));
  setAntennaCmd.EnabledAntennaPorts = enabledAntennas;
```



Copyright © 2008, Impinj, Inc.

```
  MAPI_DATA response = MAPI_SetAntennaCmd(_reader,&setAntennaCmd);

  //Check for an Invalid Command Notification
  switch (response.Type){
         case MAPI_DATA_TYPE_INVALID_COMMAND_NTF:
                {
                        MAPI_DATA_INVALID_COMMAND_NTF *invalidCommand =
(MAPI_DATA_INVALID_COMMAND_NTF *)response.Data;
                        printf("Received Invalid Command Notification while enabling antennas
%d\n", enabledAntennas);
                        printf("Reason Code = %d, Reader State = %d\n",invalidCommand->ReasonCode,
invalidCommand->State);
                        _lastMessageResult.messageType = MAPI_DATA_TYPE_INVALID_COMMAND_NTF;
                        _lastMessageResult.resultCode = invalidCommand->ReasonCode;
                        MAPI_Free(invalidCommand);
                        break;
                }
         case MAPI_DATA_TYPE_SET_ANTENNA_RSP:
                {
                        MAPI_DATA_SET_ANTENNA_RSP *setAntennaRsp;
                        setAntennaRsp = (MAPI_DATA_SET_ANTENNA_RSP *)response.Data;
                        _lastMessageResult.messageType = MAPI_DATA_TYPE_SET_ANTENNA_RSP;
                        _lastMessageResult.resultCode = setAntennaRsp->ResultCode;
                        MAPI_Free(setAntennaRsp);
                        if (_lastMessageResult.resultCode != 0){
                                printf("Could not enable antenna ports %d . Result Code:
%d\n",enabledAntennas, _lastMessageResult.resultCode);
                        }
                        else{
                                printf("Configured antenna ports: 0x%02x\n", enabledAntennas);
                        }
                        break;
                }
  }


  printf("Starting Inventory\n");
  StartInventory();
  Sleep(_inventoryDurationMillis);
  StopInventory();
  printf("Inventory finished\n");
  SetEvent(_tagsInventoriedNotification);

  _inventoryIsRunning = false;
  return 0;
}
```

Copyright © 2008, Impinj, Inc.

```
//The MAPI message processing thread
UINT DataThread (LPVOID Arg)
{

  //Create our thread synchronization objects
  _rebootNotification                                = CreateEvent(NULL, FALSE, FALSE, NULL);
  _regulatoryRegionChangeNotification = CreateEvent(NULL, FALSE, FALSE, NULL);
  _modemStoppedNotification                   = CreateEvent(NULL, FALSE, FALSE, NULL);
  _tagsInventoriedNotification         = CreateEvent(NULL, FALSE, FALSE, NULL);

  while(_reader){
          struct MAPI_DATA ReturnedData;
          ReturnedData = MAPI_GetData(_reader);

          switch (ReturnedData.Type){

                  ////////////////////////////
                  // Received BOOT notification
                  ////////////////////////////
                  case MAPI_DATA_TYPE_BOOT_MODEM_NTF:
                  {
                          struct  MAPI_DATA_BOOT_MODEM_NTF *bootModemNtf;
                          bootModemNtf = (MAPI_DATA_BOOT_MODEM_NTF *)ReturnedData.Data;

                          //save the reboot command result code
                          unsigned char rebootResultCode = bootModemNtf->BootResultCode;

                          //Free the returned data
                          MAPI_Free(bootModemNtf);

                          //result code 1 = boot progress notification- we are only interested
                          //in success or failure codes
                          if (1 == rebootResultCode){
                                  //do something here if you care about monitoring boot progress
                          }
                          else{
                                  //Boot is complete. Boot result code is stored in _rebootResultCode
                                  SetEvent(_rebootNotification);

                                  _lastMessageResult.messageType = MAPI_DATA_TYPE_BOOT_MODEM_NTF;
                                  _lastMessageResult.resultCode = rebootResultCode;

                          }
                          break;
                  }
                  ////////////////////////////////////////////////////
```


Copyright © 2008, Impinj, Inc.

```
                // Received Set Regulatory Region result notification
                ////////////////////////////////////////////////////
                case MAPI_DATA_TYPE_SET_REGULATORY_REGION_NTF:
                {
                        struct  MAPI_DATA_SET_REGULATORY_REGION_NTF *data;
                        data = (MAPI_DATA_SET_REGULATORY_REGION_NTF *)ReturnedData.Data;

                        //save the regulatory change request result code
                        _lastMessageResult.messageType = MAPI_DATA_TYPE_SET_REGULATORY_REGION_NTF;
                        _lastMessageResult.resultCode = data->ResultCode;

                        //Free the returned data
                        MAPI_Free(ReturnedData.Data);
                        SetEvent(_regulatoryRegionChangeNotification);
                        break;
                }
                ////////////////////////////////////////////////////
                // Received Tag Read Notification
                ////////////////////////////////////////////////////
                case MAPI_DATA_TYPE_INVENTORY_NTF:
                {
                        MAPI_DATA_INVENTORY_NTF  *inventoryNtf = (MAPI_DATA_INVENTORY_NTF
*)ReturnedData.Data;
                        EPC epc(inventoryNtf->Epc, inventoryNtf->EpcLen);
                        MAPI_Free(inventoryNtf);
                        _inventoriedTags.insert(epc);
                        break;
                }

                ////////////////////////////////////////////////////
                // Received a GPIO notification
                ////////////////////////////////////////////////////
                case MAPI_DATA_TYPE_GPI_ALERT_NTF:
                {
                        if (! _inventoryIsRunning){
                                MAPI_DATA_GPI_ALERT_NTF *gpiAlertNtf = (MAPI_DATA_GPI_ALERT_NTF
*)ReturnedData.Data;
                                //save our last input triggered
                                _lastInputTriggered = gpiAlertNtf->TriggeredInput;
                                MAPI_Free(gpiAlertNtf);

                                //Start the inventory timer thread
                                _hInventoryThread = AfxBeginThread(InventoryTimerThread, NULL ,
THREAD_PRIORITY_NORMAL);
                                if (! _hInventoryThread){
                                        printf("Could not create Inventory Thread\n");
                                        exit(1);
```

Copyright © 2008, Impinj, Inc.

```
                                }
                        }
                        break;
                }

                ////////////////////////////////////////////////////
                // Received a Modem Stopped Notification
                // (i.e. Inventory stopped)
                ////////////////////////////////////////////////////
                case MAPI_DATA_TYPE_MODEM_STOPPED_NTF:
                {
                        MAPI_DATA_MODEM_STOPPED_NTF *modemStoppedNtf =
(MAPI_DATA_MODEM_STOPPED_NTF *)ReturnedData.Data;
                        _lastMessageResult.messageType = MAPI_DATA_TYPE_MODEM_STOPPED_NTF;
                        _lastMessageResult.resultCode = modemStoppedNtf->ModemStopReason;
                        MAPI_Free(modemStoppedNtf);
                        SetEvent(_modemStoppedNotification);
                        break;

                }

                ////////////////////////////////////////////////////
                // Received Reader connection termination
                ////////////////////////////////////////////////////
                case MAPI_DATA_TYPE_EMPTY:
                {
                        printf("Data thread terminiating due to DLL Notification.\n");
                        //null out the reader handle- it's no longer valid
                        _reader = NULL;
                        break;
                }

                ////////////////////////////////
                // Unhandled Event Type
                ////////////////////////////////
                default:
                {
                        if (ReturnedData.Data) MAPI_Free(ReturnedData.Data);
                        break;
                }
        }
  }

  //clean up our handles

  //Notify thread is ending
```


Copyright © 2008, Impinj, Inc.

```
  SetEvent(_hDataThread->m_hThread);
  return 0;
}



////////////////////////////////////////////////////////
// Format a binary EPC into a std C string representation
////////////////////////////////////////////////////////
void FormatEPC(char *buffer, unsigned short *EPCData, unsigned int EPCLength)
{
  if (0 == EPCLength){
         strcpy(buffer,"(Blank EPC)");
  }
  else{
         char *scratch = buffer;
         unsigned short *epcIter = EPCData;
         unsigned int epcLenIter = 0;

         while (epcLenIter < EPCLength){
                //Format with dashes betwen word boundaries to improve readability
                sprintf(scratch, ((epcLenIter < EPCLength-1) ? "%04X-":"%04X"), *epcIter);
                scratch += 5;
                epcIter++;
                epcLenIter++;
         }
  }
}
```

Copyright © 2008, Impinj, Inc.

## Listing 2: Mach1_GPIO_Dock_Door.h

```
/*
 ****************************************************************************
 *                                                                          *
 *                                                                          *
 *                                                                          *
 *                                                                          *
 * (c) Copyright Impinj, Inc. 2005, 2006. All rights reserved.              *
 *                                                                          *
 ****************************************************************************
 *                                                                          *
 * This code is compatible with Mach1 2.8.0                                 *
 *                                                                          *
 ****************************************************************************
 */

#ifndef _MACH1_GPIO_DOCK_DOOR_H_
#define _MACH1_GPIO_DOCK_DOOR_H_

#define DLLEXPORT __declspec(dllimport)

#include <vector>
#include <set>


extern "C"
{

#include "mapi_extypes.h"
#include "mapi_control.h"
#include "mapi_message.h"

}


//A simple class to store EPC codes
class EPC{

 public:
        unsigned short epcData[6];
        int                             epcLength;

        EPC(unsigned short *epc, int length) : epcLength(length){
```



Copyright © 2008, Impinj, Inc.

```
                memcpy(epcData, epc, length * sizeof(unsigned short));
        }

        EPC() : epcLength(0){
                memset(epcData,0,sizeof(epcData));
        }

        EPC& operator=(const EPC &rhs){
                memcpy (epcData, rhs.epcData, rhs.epcLength * sizeof(unsigned short));
                epcLength = epcLength;
                return *this;
        }

        bool operator<(const EPC& rhs) const{
                if (rhs.epcLength > epcLength) return true;
                for (int i = 0; i < epcLength; i++){
                        if (rhs.epcData[i] > epcData[i]) return true;
                }
                return false;
        }
};

//A struct to store the last Mach1 message type and result code, if applicable
struct LastMessageResult{
 int messageType;
 unsigned char resultCode;
};


READER *ConnectReader(char *readerHost);
void BootModem();
void SetRegulatoryRegion(unsigned int regulatoryRegion );

//GPIO configuration functions
void ConfigureGPINotifications(unsigned char gpi, unsigned char config);
void ActivateGPO(unsigned char gpo, unsigned char state);

void StartInventory();
void StopInventory();

void FormatEPC(
                                char *buffer,
                                unsigned short *EPCData,
                                unsigned int EPCLength
                                );
```

Copyright © 2008, Impinj, Inc.

```
//MAPI Message Data thread function
UINT                    DataThread(LPVOID Arg);

//Inventory Timer thread function
UINT                    InventoryTimerThread(LPVOID Arg);

//The thread handle
CWinThread              *_hDataThread;
CWinThread              *_hInventoryThread;

//thread synchronization globals
HANDLE   _rebootNotification;
HANDLE   _regulatoryRegionChangeNotification;
HANDLE   _modemStoppedNotification;
HANDLE   _tagsInventoriedNotification;

READER   *_reader;

//Parameters for the running inventory round
unsigned int            _inventoryDurationMillis;

//a set that stores the unique tags found by the reader
std::set<EPC>           _inventoriedTags;

//Stores the most recent message result information
LastMessageResult       _lastMessageResult;

//Indicates if the inventory is running
bool                            _inventoryIsRunning;

//Stores the latest triggered input
unsigned char           _lastInputTriggered;
#endif
```



Copyright © 2008, Impinj, Inc.

Copyright © 2008, Impinj, Inc. All rights reserved.

## Notices

Impinj assumes no responsibility for customer product design or for infringement of patents and/or the rights of third parties, which may result from drawings, designs, or technical information provided within this application note. No representation of warranty is given and no liability is assumed by Impinj with respect to the accuracy of information provided, or use of products described within this application note. Third party products described within this document are not endorsed by Impinj and are shown for reference purposes only.

Impinj products are not designed for use in life support appliances, devices, or systems where malfunction can reasonably be expected to result in personal injury, death, property damage, or environmental damage.

Impinj, Inc.
701 N. 34th Street, Suite 300
Seattle, WA 98103
www.impinj.com