



# Driver Development Guide

Version 1.0.0.0 BETA

**Gadgetos Operating Environment**

*Copyright 2011 Thomas W. Holtquist*

[www.skewworks.com](http://www.skewworks.com)

## Table of Contents

1 About this Document .....	2
1.1 Change Log .....	2
1.2 Intended Audience .....	2
2 Driver Interfaces.....	3
2.1 IAudio .....	4
2.1.1 Example Driver .....	4
2.2 IDevice.....	14
2.2.1 Example Driver .....	14
2.3 ITime.....	21
2.3.1 Example Driver .....	21

# 1 About this Document

## 1.1 Change Log

**Nov 7<sup>th</sup>, 2011** Initial release, no changes

## 1.2 Intended Audience

This document is aimed at users with a basic knowledge of C# and the .NET Micro Framework (NETMF). Users should have a working understanding of how to implement *interfaces* and work with hardware in NETMF.

## 2 Driver Interfaces

All drivers for *Gadgets* must derive from one of the interfaces below and be compiled as a Class Library. Additionally drivers should be marked as [Serializable] and should **not** inherit from MarshalByRefObject; failing to do so will result in users and applications not being able to properly reference your driver or receive events.

Drivers are loaded at startup by *Gadgets* according to a System.XML file. This file is created at install and can be modified by the user to add new drivers from the System > Drivers screen. Once loaded, they are added to the appropriate manager inside the Kernel.

Each type of driver has a BuildID property as part of its interface; this helps you keep track of which version the user is running for update purposes when new versions of your driver become available.

## 2.1 IAudio

Drivers that control Audio playback should derive from this interface and are marked in the System.XML file as <AUDIO> nodes.

### 2.1.1 Example Driver

This example shows how to create an audio driver for the ChipworkX's built-in MP3 decoder. You will need to add references to: Skewworks.Gadgetos.Interfaces, Microsoft.SPOT.Hardware and System.IO.

```
using System;
using System.IO;
using System.Threading;

using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using Skewworks.Gadgetos;

namespace GadgetosChipworkXAudio
{
    [Serializable]
    public class ChipworkXAudio : IAudio
    {
        #region Constants

        // values
        const ushort SM_SDINew = 0x800;
        const ushort SM_RESET = 0x04;
        const ushort SC_MULT_7 = 0xE000;
        const ushort VOL_MUTE = 0xFEFE;

        // Registers
        const int SCI_MODE = 0x00;
        const int SCI_STAT = 0x01;
        const int SCI_BASS = 0x02;
        const int SCI_CLOCKF = 0x03;
        const int SCI_DECODE_TIME = 0x04;
        const int SCI_AUDATA = 0x05;
        const int SCI_HDAT0 = 0x08;
        const int SCI_HDAT1 = 0x09;
        const int SCI_VOL = 0x0B;

        #endregion

        #region Variables

        // Some GPIO pins
        private OutputPort reset;
        private InputPort DREQ;

    }
}
```

```

// Define SPI Configuration for VS1053 MP3 decoder
private SPI.Configuration dataConfig;
private SPI.Configuration cmdConfig;
private SPI spi;

private byte[] block = new byte[32];
private byte[] cmdBuffer = new byte[4];

private bool _initialized;           // Pins created successfully when true
private int _duration = -1;          // Duration of file playing (in seconds)
private int _position = -1;          // Current Play Position (in seconds)
private object _source;              // Filename or Resource Reference
private bool _busy = false;          // Playing a file when true
private bool _pause = false;         // Paused when true
private long _rawSize = -1;
private long _readSoFar = -1;
private int _Bitrate = 0;
private int _mpegLayer = 0;
private bool _stop = false;
private int _lastPos = -1;
private bool _loop = false;
private int _sample = -1;

#endregion

#region Constructor

public ChipworkXAudio()
{
    Initialize();
}

#endregion

#region Properties

public int Bitrate
{
    get { return _Bitrate; }
}

public string BuildID
{
    get { return "20111026"; }
}

public bool Busy
{
    get { return _busy; }
}

public bool CanQueue
{
    get { return false; }
}

```

```

public bool Decoding
{
    get
    {
        ushort stat = Command_Read(SCI_STAT);
        int v = stat >> 15;
        return (v == 0) ? false : true;
    }
}

public int Duration
{
    get { return _duration; }
}

public string Filename
{
    get
    {
        if (_source is string)
            return (string)_source;
        else
            return "Embedded Resource";
    }
    set { _source = value; }
}

public bool Initialized
{
    get { return _initialized; }
}

public bool Loop
{
    get { return _loop; }
    set { _loop = value; }
}

public bool Paused
{
    get { return _pause; }
    set { _pause = value; }
}

public int Position
{
    get { return _position; }
    set
    {
        throw new Exception("This method is not yet implemented");
    }
}

public string SampleRate
{
    get { return _sample.ToString() + "Kbps"; }
}
public int Volume

```

```

{
    get { return (int)((float)(VOL_MUTE - Command_Read(SCI_VOL)) / 652.78); }
    set
    {
        if (value < 0)
            value = 0;
        if (value > 100)
            value = 100;
        Command_Write(SCI_VOL, (ushort)(VOL_MUTE - (value * 652.78)));
    }
}

#endregion

#region Events

public event OnHeaderReady HeaderReady;
public event OnPlayComplete PlayComplete;
public event OnPositionChanged PositionChanged;

void OnHeaderReady(int Duration, int Bitrate, int MPEGLayer)
{
    if (HeaderReady != null)
        HeaderReady(Duration, Bitrate, MPEGLayer);
}

void OnPlayComplete()
{
    if (PlayComplete != null)
        PlayComplete();
}

void OnPositionChanged(int Position)
{
    if (PositionChanged != null)
        PositionChanged(Position);
}

#endregion

```

```

#region Public Methods

public void Initialize()
{
    if (_initialized)
        return;

    try
    {
        dataConfig = new SPI.Configuration((Cpu.Pin)15, false, 0, 0, false, true,
3000, SPI.SPI_module.SPI1);
        cmdConfig = new SPI.Configuration((Cpu.Pin)68, false, 0, 0, false, true,
3000, SPI.SPI_module.SPI1);

        spi = new SPI(cmdConfig);
        reset = new OutputPort((Cpu.Pin)31, true);
        DREQ = new InputPort((Cpu.Pin)71, false, Port.ResistorMode.PullUp);
        _initialized = true;

        Reset();
        Command_Write(SCI_MODE, SM_SDINew);
        Command_Write(SCI_CLOCKF, SC_MULT_7);
        Command_Write(SCI_VOL, 0);
    }
    catch (Exception e) { throw e; }
}

public void Play()
{
    new Thread(PlayLoop).Start();
}

public void Play(string Filename)
{
    _source = Filename;
    new Thread(PlayLoop).Start();
}

public void Play(byte[] ResourceData)
{
    _source = ResourceData;
    new Thread(PlayLoop).Start();
}

```

```

public void Reset()
{
    while (DREQ.Read() == false) ;

    Command_Write(SCI_MODE, (ushort)(Command_Read(SCI_MODE) | SM_RESET));
    Thread.Sleep(1);

    // Reset Variables
    _duration = -1;
    _mpegLayer = 0;
    _position = -1;
    _Bitrate = 0;
    _source = string.Empty;
    _rawSize = 0;
    _readSoFar = 0;
    _lastPos = -1;

    while (DREQ.Read() == false) ;
}

public void Stop()
{
    _stop = true;
}

#endregion

#region Private Methods

private ushort Command_Read(byte address)
{
    ushort temp;

    while (DREQ.Read() == false) ;

    spi.Config = cmdConfig;
    cmdBuffer[0] = 0x03;
    cmdBuffer[1] = address;
    cmdBuffer[2] = 0;
    cmdBuffer[3] = 0;

    spi.WriteRead(cmdBuffer, cmdBuffer, 2);

    temp = cmdBuffer[0];
    temp <<= 8;
    temp += cmdBuffer[1];

    return temp;
}

```

```

private void Command_Write(byte address, ushort data)
{
    while (DREQ.Read() == false) ;

    spi.Config = cmdConfig;
    cmdBuffer[0] = 0x02;
    cmdBuffer[1] = address;
    cmdBuffer[2] = (byte)(data >> 8);
    cmdBuffer[3] = (byte)data;

    spi.Write(cmdBuffer);
}

private int DurationInSeconds()
{
    // Get Layer Data
    ushort val = Command_Read(SCI_HDAT1);

    if (val == 0)
        return -1;

    ushort v = (ushort)(val << 13);
    int Layer = (byte)(v >> 14);

    v = (ushort)(val << 15);
    byte ID = (byte)(v >> 14);

    // Get Bitrate
    val = Command_Read(SCI_HDAT0);
    val = (ushort)(val >> 12);

    // Get REAL Bitrate
    switch (Layer)
    {
    case 0:
        _mpegLayer = 0;
        return -1;
    case 1:
        _mpegLayer = 3;
        switch (val)
        {
        case 1:
            if (ID == 3)
                _Bitrate = 32;
            else
                _Bitrate = 8;
            break;
        case 2:
            if (ID == 3)
                _Bitrate = 40;
            else
                _Bitrate = 16;
            break;
        case 3:
            if (ID == 3)
                _Bitrate = 48;
            else
                _Bitrate = 24;
        }
    }
}

```

```
        break;
    case 4:
        if (ID == 3)
            _Bitrate = 56;
        else
            _Bitrate = 32;
        break;
    case 5:
        if (ID == 3)
            _Bitrate = 64;
        else
            _Bitrate = 40;
        break;
    case 6:
        if (ID == 3)
            _Bitrate = 80;
        else
            _Bitrate = 48;
        break;
    case 7:
        if (ID == 3)
            _Bitrate = 96;
        else
            _Bitrate = 56;
        break;
    case 8:
        if (ID == 3)
            _Bitrate = 112;
        else
            _Bitrate = 64;
        break;
    case 9:
        if (ID == 3)
            _Bitrate = 128;
        else
            _Bitrate = 80;
        break;
    case 10:
        if (ID == 3)
            _Bitrate = 160;
        else
            _Bitrate = 96;
        break;
    case 11:
        if (ID == 3)
            _Bitrate = 192;
        else
            _Bitrate = 112;
        break;
    case 12:
        if (ID == 3)
            _Bitrate = 224;
        else
            _Bitrate = 128;
        break;
    case 13:
        if (ID == 3)
            _Bitrate = 256;
```

```

        else
            _Bitrate = 144;
        break;
    case 14:
        if (ID == 3)
            _Bitrate = 320;
        else
            _Bitrate = 16;
        break;
    }
    break;
case 2:
    _mpegLayer = 2;
    return -1;
case 3:
    _mpegLayer = 1;
    return -1;
}

int dur = (int)((((_rawSize - _readSoFar) * 8) / _Bitrate / 1000));

OnHeaderReady(dur, _Bitrate, _mpegLayer);
return dur;
}

private void PlayLoop()
{
    byte[] b;

    _busy = true;
    _stop = false;
    while (!_stop)
    {
        _readSoFar = 0;
        _lastPos = -1;

        if (_source is string)
        {
            if (File.Exists((string)_source))
            {
                FileStream iFile = new FileStream((string)_source, FileMode.Open,
FileAccess.Read);

                _rawSize = iFile.Length;
                b = new byte[2 * 1024];
                int count = iFile.Read(b, 0, b.Length);
                while (count > 0 && !_stop)
                {
                    SendData(b);
                    b = new byte[2 * 1024];
                    count = iFile.Read(b, 0, b.Length);
                }
                iFile.Close();
            }
        }
        else if (_source is byte[])
        {
            b = (byte[])_source;
            _rawSize = b.Length;

```

```

        SendData(b);
    }

    _position = _duration;
    OnPositionChanged(_position);
    Reset();
    OnPlayComplete();

    if (!Loop)
        break;
}
_busy = false;
}

private void SendData(byte[] data)
{
    int size = data.Length - data.Length % 32;

    spi.Config = dataConfig;
    for (int i = 0; i < size; i += 32)
    {
        while (DREQ.Read() == false)
            Thread.Sleep(1);    // wait till done

        // Update Information
        if (_duration == -1)
            _duration = DurationInSeconds();
        _position = Command_Read(SCI_DECODE_TIME);

        if (_position != _lastPos)
        {
            OnPositionChanged(_position);
            _lastPos = _position;
        }

        // We need to return to data config
        spi.Config = dataConfig;

        Array.Copy(data, i, block, 0, 32);

        spi.Write(block);
        _readSoFar += 32;    // We need to know how many bytes we've read to
        calculate header size to get duration

        while (_pause)
            Thread.Sleep(1);    // Wait while paused
    }
}

#endregion
}
}
}

```

## 2.2 IDevice

The IDevice interface is meant for drivers that control HIDs (Human Interface Devices) and are marked in the System.XML file as <HID> nodes.

This type of driver allows for two-way communication. It can communicate to the user through a series of events and it can receive data from the user by a simple SendData call.

### 2.2.1 Example Driver

This example shows how to create a HID driver for the Emulator. You will need to add references to: Skewworks.Gadgetos.Interfaces, and Microsoft.SPOT.Hardware.

```
using System;

using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using Skewworks.Gadgetos;

namespace Skewworks.Gadgetos.Hardware
{
    [Serializable]
    public class EmulatorHID : IDevice
    {
        #region Variables

        private InterruptPort pipUp, pipRight, pipDown, pipLeft, pipStart;

        #endregion

        #region Constructor

        public EmulatorHID()
        {
            try
            {
                pipDown = new InterruptPort((Cpu.Pin)4, true, Port.ResistorMode.PullUp,
Port.InterruptMode.InterruptEdgeBoth);
                pipDown.OnInterrupt += new NativeEventHandler(pipDown_OnInterrupt);
            }
            catch (Exception) { }

            try
            {
                pipLeft = new InterruptPort((Cpu.Pin)0, true, Port.ResistorMode.PullUp,
Port.InterruptMode.InterruptEdgeBoth);
                pipLeft.OnInterrupt += new NativeEventHandler(pipLeft_OnInterrupt);
            }
            catch (Exception) { }
        }
    }
}
```

```

        try
        {
            pipRight = new InterruptPort((Cpu.Pin)1, true, Port.ResistorMode.PullUp,
Port.InterruptMode.InterruptEdgeBoth);
            pipRight.OnInterrupt += new NativeEventHandler(pipRight_OnInterrupt);
        }
        catch (Exception) { }

        try
        {
            pipStart = new InterruptPort((Cpu.Pin)3, true, Port.ResistorMode.PullUp,
Port.InterruptMode.InterruptEdgeBoth);
            pipStart.OnInterrupt += new NativeEventHandler(pipStart_OnInterrupt);
        }
        catch (Exception) { }

        try
        {
            pipUp = new InterruptPort((Cpu.Pin)2, true, Port.ResistorMode.PullUp,
Port.InterruptMode.InterruptEdgeBoth);
            pipUp.OnInterrupt += new NativeEventHandler(pipUp_OnInterrupt);
        }
        catch (Exception) { }
    }

#endregion

#region Properties

public string BuildID
{
    get { return "20111026"; }
}

public int JoystickID
{
    get { return -1; }
    set { }
}

public Directions JoystickMajorDirection
{
    get { return Directions.None; }
}

public int JoystickX
{
    get { return -1; }
}

public int JoystickY
{
    get { return -1; }
}

```

```

public virtual bool UsesMousePointer
{
    get { return false; }
}

public bool UsesJoystick
{
    get { return false; }
}

#endregion

#region Events

public event OnHIDAction HIDAction;
protected virtual void OnHIDAction(object sender, HIDInputAction type, int
value1, int value2)
{
    if (HIDAction != null)
        HIDAction(sender, type, value1, value2);
}

public event OnBluePressed BluePressed;
protected virtual void OnBluePressed()
{
    if (BluePressed != null)
        BluePressed();
}

public event OnRedPressed RedPressed;
protected virtual void OnRedPressed()
{
    if (RedPressed != null)
        RedPressed();
}

public event OnGreenPressed GreenPressed;
protected virtual void OnGreenPressed()
{
    if (GreenPressed != null)
        GreenPressed();
}

public event OnYellowPressed YellowPressed;
protected virtual void OnYellowPressed()
{
    if (YellowPressed != null)
        YellowPressed();
}

public event OnSelectPressed SelectPressed;
protected virtual void OnSelectPressed()
{
    if (SelectPressed != null)
        SelectPressed();
}

```

```

public event OnStartPressed StartPressed;
protected virtual void OnStartPressed()
{
    if (StartPressed != null)
        StartPressed();
}

public event OnUpPressed UpPressed;
protected virtual void OnUpPressed()
{
    if (UpPressed != null)
        UpPressed();
}

public event OnDownPressed DownPressed;
protected virtual void OnDownPressed()
{
    if (DownPressed != null)
        DownPressed();
}

public event OnLeftPressed LeftPressed;
protected virtual void OnLeftPressed()
{
    if (LeftPressed != null)
        LeftPressed();
}

public event OnRightPressed RightPressed;
protected virtual void OnRightPressed()
{
    if (RightPressed != null)
        RightPressed();
}

public event OnBlueReleased BlueReleased;
protected virtual void OnBlueReleased()
{
    if (BlueReleased != null)
        BlueReleased();
}

public event OnRedReleased RedReleased;
protected virtual void OnRedReleased()
{
    if (RedReleased != null)
        RedReleased();
}

public event OnGreenReleased GreenReleased;
protected virtual void OnGreenReleased()
{
    if (GreenReleased != null)
        GreenReleased();
}

public event OnYellowReleased YellowReleased;

```

```

protected virtual void OnYellowReleased()
{
    if (YellowReleased != null)
        YellowReleased();
}

public event OnSelectReleased SelectReleased;
protected virtual void OnSelectReleased()
{
    if (SelectReleased != null)
        SelectReleased();
}

public event OnStartReleased StartReleased;
protected virtual void OnStartReleased()
{
    if (StartReleased != null)
        StartReleased();
}

public event OnUpReleased UpReleased;
protected virtual void OnUpReleased()
{
    if (UpReleased != null)
        UpReleased();
}

public event OnDownReleased DownReleased;
protected virtual void OnDownReleased()
{
    if (DownReleased != null)
        DownReleased();
}

public event OnLeftReleased LeftReleased;
protected virtual void OnLeftReleased()
{
    if (LeftReleased != null)
        LeftReleased();
}

public event OnRightReleased RightReleased;
protected virtual void OnRightReleased()
{
    if (RightReleased != null)
        RightReleased();
}

public event OnJoystickButtonPressed JoystickButtonPressed;
protected virtual void OnJoystickButtonPressed(int ID, int ButtonID)
{
    if (JoystickButtonPressed != null)
        JoystickButtonPressed(ID, ButtonID);
}

```

```

public event OnJoystickButtonReleased JoystickButtonReleased;
protected virtual void OnJoystickButtonReleased(int ID, int ButtonID)
{
    if (JoystickButtonReleased != null)
        JoystickButtonReleased(ID, ButtonID);
}

public event OnJoystickMajorDirectionChanged JoystickMajorDirectionChanged;
protected virtual void OnJoystickMajorDirectionChanged(int ID, Directions e)
{
    if (JoystickMajorDirectionChanged != null)
        JoystickMajorDirectionChanged(ID, e);
}

public event OnJoystickMoved JoystickMoved;
protected virtual void OnJoystickMoved(int ID, point e)
{
    if (JoystickMoved != null)
        JoystickMoved(ID, e);
}

#endregion

#region Interrupt Events

private void pipDown_OnInterrupt(uint pin, uint state, DateTime time)
{
    if (!pipDown.Read())
        OnDownPressed();
    else
        OnDownReleased();
}

private void pipLeft_OnInterrupt(uint pin, uint state, DateTime time)
{
    if (!pipLeft.Read())
        OnLeftPressed();
    else
        OnLeftReleased();
}

private void pipRight_OnInterrupt(uint pin, uint state, DateTime time)
{
    if (!pipRight.Read())
        OnRightPressed();
    else
        OnRightReleased();
}

private void pipStart_OnInterrupt(uint pin, uint state, DateTime time)
{
    if (!pipStart.Read())
        OnStartPressed();
    else
        OnStartReleased();
}

```

```
private void pipUp_OnInterrupt(uint pin, uint state, DateTime time)
{
    if (!pipUp.Read())
        OnUpPressed();
    else
        OnUpReleased();
}

#endregion

#region Public Methods

public point JoystickInRegion(int X, int Y, int Width, int Height)
{
    return new point(-1, -1);
}

public void SendMessage(int MessageID, int[] Parameters)
{
}

#endregion

}
}
```

## 2.3 ITime

ITime provides the simplest driver interface and allows Gadgetos to get the current time from a Real Time Clock (RTC) or other source interface and are marked in the System.XML file as <TIME> nodes.

### 2.3.1 Example Driver

This example shows how to create a time driver for the ChipworkX's built-in RTC. You will need to add references to: Skewworks.Gadgetos.Interfaces, GHIElectronics.NETMF.Hardware, and Microsoft.SPOT.Time.

```
using System;
using System.Globalization;

using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.Time;

using GHIElectronics.NETMF.Hardware;

using Skewworks.Gadgetos;

namespace GadgetosChipworkXTime
{
    public class ChipworkXTime : MarshalByRefObject, ITime
    {
        #region Properties

        public string BuildID
        {
            get { return "20111026"; }
        }

        public DateTime DateTime
        {
            get
            {
                Utility.SetLocalTime(RealTimeClock.GetTime());
                return RealTimeClock.GetTime();
            }
            set
            {
                Debug.Print("Setting time to: " + value);
                RealTimeClock.SetTime(value);
                Debug.Print("Time is: " + RealTimeClock.GetTime().ToString());
                Utility.SetLocalTime(value);
            }
        }
    }
}
```

```
public int TimeZone
{
    get
    {
        return 0;
    }
    set
    {
        TimeService.SetTimeZoneOffset(value);
    }
}

#endregion

}
```