Gizmondo Developer's Guide

V1.5

Contents

Executive Summary

This document specifies the interfaces that a games development company may use to interface to and create content for the Gizmondo device.

References

"Gizmondo Installation 1.1.pdf", Gizmondo Studios, Oct 2004.

Gizmondo Digital Signatures 1.0.pdf, 21/01/2005

Gizmondo SD Card Protection.pdf, 03/05/2005

"GPS Specification.pdf", Plextek Limited, 25/5/2004.

"MIDI Implementation Chart.pdf", Micronas GmbH, 9/12/2003.

"Supported Midi Banks and Instruments.pdf", Micronas GmbH.

Gizmondo Certification Requirements, v0.3, Gizmondo Studios, May 2005.

"Tech Conference v2.ppt", Gizmondo Studios, Dec 2004.

1. Introduction

1.1. Specification Overview

The Gizmondo is a high performance electronic gaming device that also features GPS and GPRS to provide location based services. The design aim is for market leading games performance and multimedia functions combined with small size in order to make the Gizmondo device attractive to the 10 to 35 year age group.

1.2. Hardware Features

The Gizmondo device features the following hardware:

Samsung 400MHz S3C2440 ARM9 CPU
64MB Flash memory and 64MB mobile SDRAM
SD/MMC card slot
2.8" QVGA TFT display
nVidia GoForce 4500 3D Media Processor
48kHz 16-bit stereo audio, plus MIDI/MP3 DSP
VGA camera
Bluetooth
GPS
GSM + GPRS
USB Desktop PC connectivity using Microsoft® ActiveSync

1.3. Software Features

The Gizmondo device runs Microsoft® Windows CE version 4.2 and supports:

Application development using eMbedded Visual C++ 4.0 SP2 with dedicated SDK
Gizmondo specific user shell (GTShell)

Developer sample code to allow games to co-operate with the Gizmondo shell 3D graphics support through OpenGL-ES version 1.0

All software running on the Gizmondo device must be digitally signed (Section 3) – contact Gizmondo for full details.

1.4. Developer Samples

This is a set of example applications (with full source code), that access areas of functionality useful when writing 3rd party games/applications on the Gizmondo unit.

This samples allows OEM information about the status of the Gizmondo device to be easily examined by an application (such as a gaming engine). Other functionality includes information flow back to the gaming unit itself, if the game needs to signal a specific task has been completed.

Some of these areas of functionally are outside of the scope of the normal CE4.2 model or are simply included here as an aid to 3rd parties integrating applications on the Gizmondo device.

Example Code

Bluetooth (via Virtual com port)
Camera
Direct Framebuffer Access (through DirectDraw)
GPS
Reading Keys
MIDI playback
Movie Playback (via DirectShow)
Handling System Messages
Vibrator Usage

2. Programming Guide

2.1. Memory and File Systems

All software on the Gizmondo device executes from the 64MB SDRAM. Approximately 20MB is reserved for the Windows CE operating system, leaving 44 MB for RAM file system and OS memory. The default split is 30%:70% and can be adjusted at run time using the Windows CE **GetSystemMemoryDivision** and **GetSystemMemoryDivision** functions.

2.2. Flash Disk

Internal non-volatile storage is provided by a Flash disk and is accessed as the "\Flash Disk" folder. This is a valuable shared resource and should only be used by games to save icon and game state data. Game information is located in the following folder:

```
\Flash disk
      \MyGames
            \<Game Title>
                  <Game Title>.gif  - icon
```

The flash disk also holds the non-volatile registry hives. Registry changes are not automatically flushed through to the hives, therefore it is recommended that **RegFlushKey** is used to commit changes to flash.

2.3. SD/MMC Cards

Individual games and game collections will be provided on SD or MMC cards. SD cards are preferred since they can be copy protected[1] and are faster since the have a 4-bit data path rather than a single bit. Both type of cards should be read-only and are accessed through the "SD Card" folder.

2.3.1. Card Layout

The layout of the SD/MMC card should be as follows:

```
\SD Card
      autorun.exe          - launched by GTShell
      \system              - graphics drivers
            nvddi.dll
            gfsdk.dll
            libgles_cm.dll
```

The system folder is only required for 3D games and should contain the graphics drivers that the game was originally developed against. GTShell will only install drivers from SD/MMC card if the drivers are newer than the drivers already in use.

2.3.2. Card Insertion / Removal

Two custom named events are signalled on card insertion and removal. These are defined in **oeminc.h** as follows:

```
#define SDMMC_INSERT_EVENT    TEXT("SDMMC_INSERT_EVENT")
#define SDMMC_REMOVE_EVENT    TEXT("SDMMC_REMOVE_EVENT")
```

[1] The SD Card copy protection scheme is yet to be implemented.

The insertion event is signalled after the card driver is successfully loaded and the removal event is signalled **before** the driver is unloaded. The removal event as it gives an early warning and is must faster than waiting for notification through FATFS.

Applications running from SD/MMC cards are expected to monitor card removal and insertion events and take appropriate action to prevent locking the system.

2.4. Display

The display is 16-bpp with QVGA resolution and can be accessed through several drivers:

Windows GDI
The display driver supports the standard Windows GDI API and features 2D acceleration for line drawing and BLTs. The games SDK provides an extension to provide direct and exclusive access to the frame buffer if required.

Direct Draw
Direct Draw is now supported[2] on 3D boards only.

OpenGL-ES
Developers wishing to use OpenGL ES for 3D graphics development will need to sign a developer's agreement. Contact Gizmondo for further information.

2.5. Backlight

The backlight supports 32 brightness levels from 0 (off) to 31 (maximum). Two levels are defined in the registry for On and Dim level. Refer to the power management section (Section 6) for how these are used.

The On and Dim levels are defined through the following registry values:

```
; Backlight setting
[HKEY_CURRENT_USER\ControlPanel\Backlight]
   "OnLevel"=dword:18
   "DimLevel"=dword:8
```

An application can change the backlight levels as follows:
Modify the registry value(s)
Signal the "**BackLightChangeEvent**" named event

The shell uses four backlight levels: 1, 11, 21 and 31. Applications are recommended to use these for consistency.

2.6. Keypad

The keypad is divided into several sections:

- Four-way joypad to the left of the screen
- Four action buttons to the right of the screen
- Two shoulder buttons
- Five piano keys across the top of the screen

Developers are encouraged to keep to the programming conventions used by the shell:

[2] Requires build 1.0.0036 or later.

Key	VK Code	Usage	Description
Joypad North	VK_UP	Move Up	
Joypad East	VK_RIGHT	Move right	
Joypad South	VK_DOWN	Move down	
Joypad West	VK_LEFT	Move left	
Action North	VK_LCONTROL	Decline	
Action East	VK_SPACE	Rewind	
Action South	VK_RETURN	Accept	
Action West	VK_LSHIFT	Forward	
Left Shoulder	VK_TAB		
Right Shoulder	VK_ESCAPE	Menu	
Piano 1 (left)	VK_F1	Home	Navigates to main screen
Piano 2	VK_F2	Volume	Displays volume control
Piano 3	VK_F3	Backlight	Displays backlight control
Piano 4	VK_F4	Alarm	Signals emergency alarm
Piano 5 (right)	VK_F11	Power	Invokes power menu

Refer to the **Keys** developer sample project for more details.

2.6.1. Piano Key Handling

Games are required to handle piano key presses as follows:

Home - return to game main menu screen
Volume - in-game volume up/down/mute control
Backlight - in-game brightness control
Alarm - no action
Power - drop to shell's power menu (Section 6.1.4)

Refer to ref 0 for further details.

2.7. Audio System

The audio hardware supports Wave (PCM) and MIDI playback. Support for hardware MP3 playback is also proposed. There is no audio input facility. Audio is routed to either the internal mono-speaker or to the 3.5 mm stereo headphone jack and is switched automatically.

More advanced audio formats are supported through Windows CE multimedia and DirectShow:

Microsoft GSM 6.10 Audio (GSM 610)
Microsoft CCITT G.711 A-Law and u-Law
Microsoft Adaptive Differential Pulse Code Modulation (MS ADPCM)
Interactive Multimedia Association Adaptive Differential Pulse Code Modulation (IMA ADPCM)
Microsoft MPEG-1 Layer 1; Microsoft MPEG-1 Layer 2
Windows Media Audio (WMA) v2, v7, v8, v9 (including Windows Media Audio 9 Voice)
Fraunhofer MPEG-1 Layer 3 (MP3)

Hardware playback support for MP3 using the Micronas MAS35x5G is available from on the Gizmondo terminal version 3D V3 onwards.

2.7.1. Wave Audio

The driver supports multiple simultaneous PCM streams. These are re-sampled and combined into a single 16-bit 48kHz stereo stream and fed to the stereo DAC. The driver supports the following PCM formats:

8 / 16 bit
Mono / stereo
8, 16, 32, 48, 11.025, 22.05, 44.1 kbit/s

The **PlaySound** and **waveOutOpen** API sets are supported. Direct Sound is not supported.

Volume is controlled by the **waveOutSetVolume** API. The master volume is controlled by passing 0 as the *hwo* parameter. Otherwise only the stream specified by *hwo* is affected.

2.7.2. MASG Device (MIDI/MP3)

The Micronas MAS35x5G synthesiser supports MIDI and MP3 playback. MIDI support includes the GM1, GML and SP-MIDI formats (ref C & ref D). The device only supports a single MP3 or MIDI format 0 stream and the maximum number of MIDI voices is 33.

The MASG driver is controlled by the following definitions in OEMINC.H:

```
IOCTL_OEM_MASG_START          - start MASG playback
IOCTL_OEM_MASG_WRITE          - pass next MIDI/MP3 buffer to MASG
IOCTL_OEM_MASG_STOP           - stops MASG playback
IOCTL_OEM_MASG_PAUSE          - pauses MASG playback
IOCTL_OEM_MASG_RESUME         - resumes MASG playback
IOCTL_OEM_MASG_STATE          - returns current MASG state
IOCTL_OEM_MASG_GETPOSITION    - returns current MASG position (bytes)
IOCTL_OEM_MASG_GETVOLUME      - returns current MASG volume
IOCTL_OEM_MASG_SETVOLUME      - sets MASG volume
IOCTL_OEM_MASG_HAS_MP3        - returns TRUE if MASG supports MP3 playback

// MASG hardware states

typedef enum
{
      MASG_UNKNOWN = -1,      // unknown state
      MASG_OFF = 0,           // hardware is off
      MASG_DSPON,             // clocks and DSP are on
      MASG_RUN,               // DSP is running
      MASG_STARTING,          // playback is starting
      MASG_PLAYING,           // playback is in progress
      MASG_PAUSED,            // playback is paused
      MASG_WAITING,           // waiting for completion
      MASG_DONE,              // playback is complete
}
MASG_STATE, *PMASG_STATE;
```

The MASG sample application illustrates how to play and control a MIDI or MP3 file through a set of MASG control functions which wrap the IOCTL codes above. The key function is

```
BOOL MasgStart (PUCHAR pBuf, DWORD cbBuf, PHANDLE phevCallback)

Parameters:

pBuf        - pointer to buffer containing start of MIDI/MP3 file
cbBuf       - size of MIDI/MP3 file buffer (bytes)
```

phevCallback - pointer to event to be signalled by MASG driver on completion.

After calling **MasgPlay** the sample application calls **MasgWrite** to pass the next buffer. Further buffers are passed each time the completion event is signalled until the device status changes to MASG_DONE.

Due to hardware limitations MIDI playback cannot be resumed after the terminal wakes from suspend (the MIDI synthesiser is powered down in suspend and there is no mechanism to resume playback from a known position in a file). Therefore, on wakeup the MASG driver simply signals completion of the previously playing MIDI file.

Applications using MIDI must be prepared to handle this eventuality. This limitation does not apply to MP3 playback.

Refer to the **MASG** developer sample project for more details.

2.8. Camera

The Camera can be accessed through the "CAM1:" stream driver. This supports the following modes of operation:

RGB preview mode (sizable in 8 pixel increments to 640x480)
YUV (4:2:0) Capture mode (640x480)

Both these modes are supported in HW, no software conversion is performed. The camera captures at 20fps in preview mode to DMA buffers.

Refer to the **Camera** developer sample project for more details.

2.9. Bluetooth

The Bluetooth module can be accessed using the standard CE APIs, but additional functions are provided to control the Bluetooth power state. To save power, it is assumed that Bluetooth will remain switched off until enabled on a game by game basis. Bluetooth enabled games should switch the module off on exit.

To determine if the Bluetooth is enabled the shell should be integrated. Example code illustrates how to switch the Bluetooth module power.

The system-wide GT_STATUS message is sent whenever the Bluetooth power state is changed.

2.10. GPS

The GPS1: driver should be used to read the units current location.

This should be opened with read only access.

See **EDT02001 GPS Spec.pdf** for details about driver usage and refer to the **GPS** developer sample project for more details.

2.11. GSM and GPRS

Applications use the GPRS_CONNECT to create an Internet connection and can use GPRS_STATUS and GT_STATUS to monitor the connection.

2.12. USB Device

The mini USB Type B connector is used to provide an ActiveSync connection with a desktop PC. A custom `wceusbsh.inf` file is supplied to match the Gizmondo's USB ID combination to the desktop ActiveSync application.

Refer to the Microsoft documentation for further details.

2.13. Vibrator

The vibrator can be access through the driver "VIB1:" using stream DeviceIoControl calls.

The state of the vibrator can be switched (off, on or pulsing). The on and off pulse cycle times are independently adjustable in 125ms units. Pulse Meta cycles are supported as shown by Figure 2.1

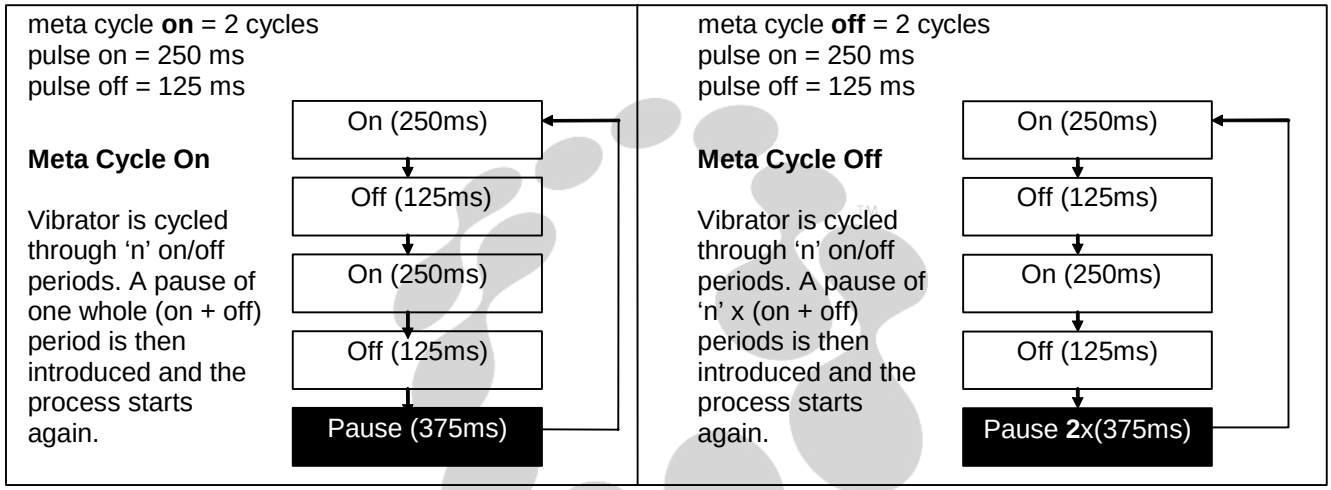


Figure 2.1 LED Meta Cycle On and Meta Cycle Off Examples

Meta cycles 'off' and 'on' cannot be mixed. Meta cycle 'on' is used if both are supplied. If no Meta cycle is required then both settings should be set to zero. Meta cycles are only used in the pulse Vibrator state. All on and off times are rounded down to a multiple of 125ms - if both are zero, the vibrator state is not changed.

Refer to the **Vibrator** developer sample project for more details.

2.14. Battery

The battery and charger status is polled by the power manager every 10 seconds.

2.14.1. Shell

The shell accesses the status through the **GetSystemPowerStatusEx2** function with *fUpdate* FALSE to prevent unnecessary sampling.

Charging starts whenever the battery voltage drops below 4.1 volts and continues until the battery is full (~ 4.2V). There is a hardware timeout of 5 hours. If the battery has not reached full charge within this period than the charger assumes the battery is faulty and switches off.

However, if the unit is in use whilst charging, then the charge entering the battery is much lower than normal and the 5 hour timeout can be exceeded without the charge completing. In this case the charger switches off and the unit runs off battery unit the battery is exhausted. **The only way to clear the fault condition is for the user to remove and re-insert the charger**.

The shell detects the battery and charger states as follows:

State	ACLineStatus	BatteryFlag	BatteryVoltage	BatteryLifePercent
Battery power	Offline (0)	High (1), Low (2) or Critical (4)	mV level	0 to 100 (%)
Charging	Online (1)	Charging (8)	mV level	Unknown (255)
Charger fault	Unknown (255)	High (1), Low (2) or Critical (4)	mV level	0 to 100 (%)

2.14.2. Applications

To applications from having to poll the power status, the shell sends the following system messages (Section 5)

```
LOWBATTERY_MSG
CRITICALBATTERY_MSG
CHARGERFAULT_MSG
```

Applications should handle the charger fault condition (timeout or temperature) by asking the user to remove (and reinsert) the charger.

3. Security

Security is provided through three main mechanisms:

- Windows CE trusted environment
- SD Card security
- Digital Rights Management (DRM)

3.1. Windows CE Trusted Environment

The Gizmondo device operates in the Windows CE trusted security mode. All modules (EXE and DLL) must be digitally signed run on the device. Unsigned modules will fail to load.

Four signing levels are implemented:

- Developer run mode
- Developer trusted mode
- User run mode
- User trusted mode

In run mode, the application is trusted to run, but is prevented from making any privileged function calls In trusted mode the application is fully trusted to perform any operation. The final trust mode of a DLL, including the 3D graphics libraries is determined by the following table.

Executable Trust	DLL Trust	Final DLL Trust
Run	Run	Run
Run	Trust	Run
Trust	Run	DLL fails to load
Trust	Trust	Trust

Distribution of developer and user keys is managed by Gizmondo Studios.

3.1.1. User Modes

By default all Gizmondo units will only be able to run applications signed with either of the user mode keys. It is expected that Gizmondo will remain the sole signing authority for user mode keys and that signing will form part of the final game approvals process.

3.1.2. Developer Modes

Developers units will also have the ability to run applications signed with the developers keys. These keys will be distributed to developers by Gizmondo and signing will need to be incorporated into the developers build processes. Further instructions will be distributed with the developers keys.

3.2. SD Card Security

The SD card security scheme is designed to protect executable files on the SD card. The executable and all DLLs must first be signed as above and then the executable is encrypted by Gizmondo. The encryption process follows the SD security specification of the SD Card Association.

The SDRun sample application illustrates how an application can launch an encrypted file from SD card.

3.3. DRM

The Microsoft Digital Rights Management scheme is designed to protect Windows Media files (WMA/WMV) transferred to the Gizmondo device. Therefore DRM is not within the scope of this document.

4. Keyboard SIP support

Applications can request the shell to draw a software input panel over part of the screen, this is useful as it allows a universal input panel for all applications and also includes predictive text support. Note the skin must support this feature else no SIP will be shown.

The SIP is invoked by sending a WM_COPYDATA message with a data structure:

Byte Offset	Value
0	0x90
1	Reserved, set to 0
2	Settings Bit 0 – No Snap to centre 1 – AutoCase on 2 – Predictive Text on 3 – Reserved 4 – Reserved 5 – Reserved 6 – Reserved 7 – Reserved
3-7	Reserved, set to 0
8-	WCHAR buffer containing initial text to populate control with

The SIP component then feeds back inputted characters via

WM_CHAR	SIP Characters
WM_KEYDOWN/KEYUP	VK_DELETE/VK_RIGHT/VK_LEFT

These return messages are sent to the whichever window was foreground when the Shell took control.

5. System Messages

The following messages are sent on a system wide basis from the shell to any waiting applications or games:

<table>
<tr><th>Message Name</th><th>Purpose</th></tr>
<tr><td>LOWBATERY_MSG</td><td>Sent when battery reaches low state</td></tr>
<tr><td>CRITICALBATERY_MSG</td><td>Sent when battery reaches critical state

The system will turn off 30 seconds after this point, games should save any information as the system will be shut down after this time.</td></tr>
<tr><td>CHARGERFAULT_MSG</td><td>Sent when a charger fault is detected.</td></tr>
<tr><td>SMS_MSG</td><td>Sent when an SMS/MMS or email is received by the system</td></tr>
<tr><td>GPRS_STATUS</td><td>Sent when GPRS is connected, the following status information is also given:

wParam

See RASCONNSTATE in CE help for more information
<table>
<tr><td>RASCS_Connected</td><td>8092</td></tr>
<tr><td>RASCS_Disconnected</td><td>8093</td></tr>
</table>
lParam

Indicates the type of connections already active (multiple connections can be active
<table>
<tr><th>Name</th><th>Bit</th></tr>
<tr><td>Internet/MMS</td><td>0</td></tr>
<tr><td>Reserved</td><td>1</td></tr>
<tr><td>Active Sync</td><td>2</td></tr>
</table></td></tr>
<tr><td>GT_STATUS</td><td>This is sent when a state below changes or when the GT_GETSTATUS is received

wParam
<table>
<tr><th>Bit</th><th>Meaning</th></tr>
<tr><td>0</td><td>Flight Mode (all radios off, GSM, GPS, Bluetooth)

0 = Flight mode Off
1 = Flight mode On</td></tr>
<tr><td>1</td><td>GPS

0 = GPS Off
1 = GPS On</td></tr>
<tr><td>2</td><td>GSM

0 = GSM Off
1 = GSM On</td></tr>
<tr><td>3</td><td>Bluetooth</td></tr>
</table></td></tr>
</table>

		0 = BT Off 1 = BT On
	4	Internet/MMS GPRS connected
	5	Reserved
	6	Active Sync Connected

The following messages can be sent from an application to the shell, where "Message Name" is used with **RegisterWindowMessage** to generate the windows message broadcast:

Message Name	Purpose
GPRS_CONNECT	Connect to GRPS. wParam Indicates the type of connection (multiple connection can be active) None (disconnect from MMS/Internet) 0 Internet 1 Reserved 2
GT_GETSTATUS	Send this to the shell to receive the latest information on the shells states
SWITCHTOSHELL_MSG	Send this to the shell to jump to a shell function The shell will automatically switch to full screen and revert back to the game afterwards. wParam Indicates the shell function: Power menu 0 Read message (SMS/MMS/email) 1
BT_MSG	Enable/Disable Bluetooth wParam Disable0 Enable 1 If flight mode is enable this command will be ignored. This command is also ignored if the shell setting to permanently disable BT is set.

Refer to the **SysMessages** developer sample project for more details.

6. Power Management

The Gizmondo terminal is power managed to maximise battery life. Games are required to cooperate with the power management system and in particular must be aware of the Housekeeping power state, in order to cleanly handle the suspend and resume sequence.

6.1.1. Power States

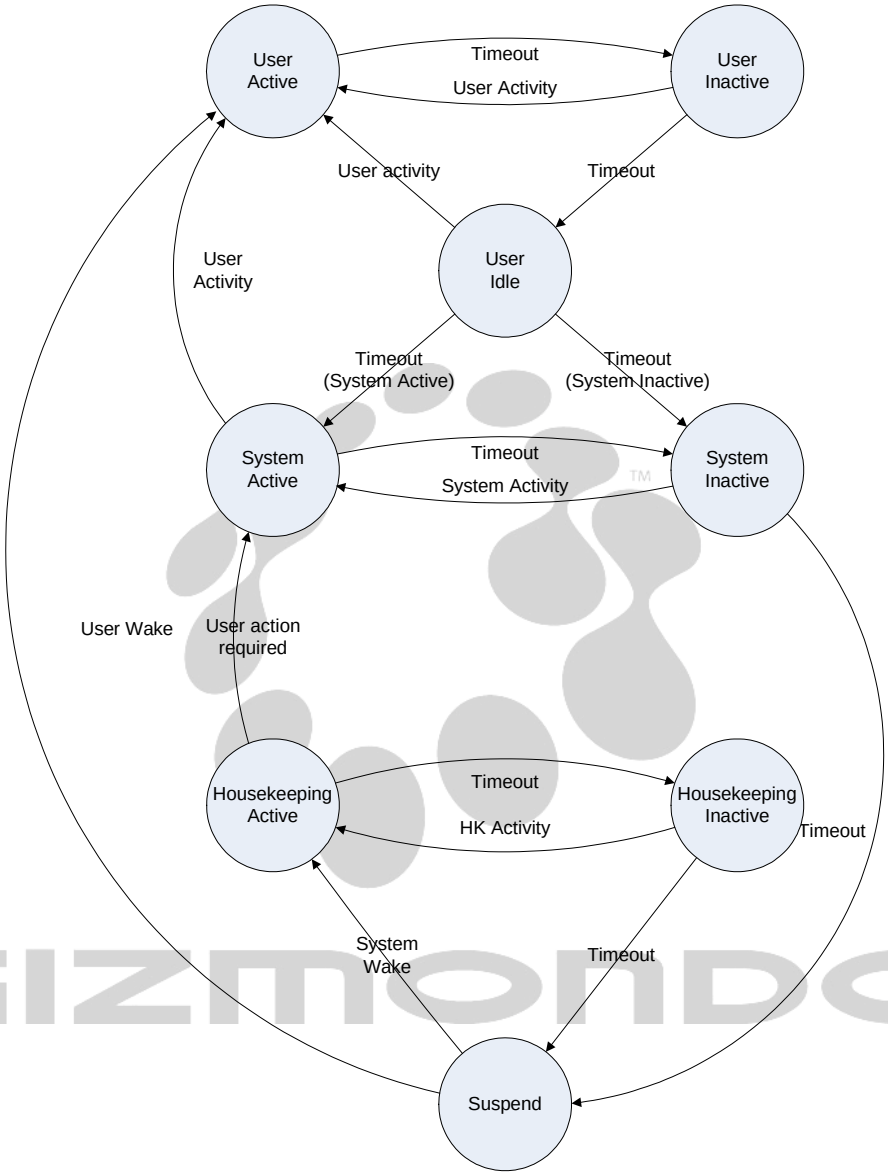
The following power states and default timeouts are defined:

System State	Power Manager State	Timeout (battery)	Timeout (charger)	Description
On	User Active	10 sec	10 sec	Full on Radio states determined by flight-mode setting.
	User Inactive	5 sec	5 sec	As User Active
User Idle	User Idle	2 min	2 min	Backlight dim. Otherwise as User Inactive
System Idle	System Active	10 sec	10 sec	Backlight off. Otherwise as User Idle
	System Inactive	1 min	1 min	As System Inactive
Housekeeping	H/K Active	10 sec	10 sec	Display and backlight off, unregistered processes forced to pause. From the user point of view, the unit appears to be in suspend.
	H/K Inactive	30 sec	30 sec	As Housekeeping Active
Suspend	Suspend	5 days	5 days	Inactive. CPU in sleep state, SDRAM in self-refresh.
Off	Off	N/A	N/A	All off except battery backed RTC.

Games developers should note the very short timeouts, especially between the On and User Idle states. Dropping from On to User Idle will cause the backlight to dim and from User Idle to System idle will cause the backlight to be switched off.

6.1.2. Power State Transitions

The following diagram illustrates typical power state transitions. Reductions in power state are caused by timeouts. Increases in power state are caused by user or system activity. Note that the **housekeeping** state can only be entered from the **suspend** state- It is skipped 'on the way down' when transitioning from **System Inactive** to **Suspend**.



6.1.3. Power Notifications and Events

Applications can use the Windows CE notification system to be informed of power state changes.

Reductions in power state can be inhibited by simulating user or system activity within the ten seconds timeout period. Three named events can be used:

Required State	Event Name	Effect
On (User Active)	"PowerManager/ActivityTimer/UserActivity"	Keeps system full on
System Idle (System Active)	"PowerManager/ActivityTimer/SystemActivity"	Inhibits suspend
Housekeeping	"PowerManager/ActivityTimer/HousekeepingActivity"	Remain in the

		Housekeeping state (Usually only for device drivers)

It is strongly recommended that 3D applications prevent system idle timeouts when OpenGL-ES is active. This is because the application must issue **glFinish()** before the display controller is powered down. If the power manager puts the system into suspend without the application first calling **glFinish()** then the graphics system can be unusable after wake from suspend.

6.1.4. Power Button Handling

Games are required to handle the power button as follows:

- Pause the game & call **glFinish()**
- Notify the shell using SWITCHTOSHELL_MSG with appropriate wParam value
- The game will lose focus and shell will display the power menu
- The game will regain focus when the user has finished with the power menu

6.1.5. Housekeeping State

As described above, the Gizmondo has a Housekeeping power state that allows the device to transparently process system events while appearing to the user to be in suspend mode.

Since the unit must *appear* to be in suspend, it is important that games and 3rd party applications do not continue running, play sounds or inadvertently boost the system power state. The Power Manager driver will ensure that this does not happen by automatically pausing **all** unregistered processes when it switches to the housekeeping mode.

However, this 'forced pause' is intended to be a mechanism for supporting legacy applications. New Gizmondo applications should register themselves as 'Housekeeping Aware' and independently handle the housekeeping state to avoid being forced to pause in this way. Typically, this will involve the following steps:

- At start-up, register as a system process
- Monitor system power state transitions
- Upon entering the Housekeeping state, cleanly pause gameplay, music playback, etc and ignore keyboard events
- Upon leaving the Housekeeping state, resume anything that was paused

The rest of this section includes sample code to perform these tasks

6.1.5.1. Registering as a System Process

Applications may register themselves as 'housekeeping aware' by opening a handle to the registration service, 'REG1:' and issuing an IOCTL command a shown below. Applications should de-register before terminating.

```
HANDLE hRegSvc;
BOOL   bRet;
TCHAR  g_szSampleProc[] = { TEXT("SampleProcess.exe") };
DWORD  dwBytes,dwCount;

hRegSvc = CreateFile(_T("REG1:"),
                     GENERIC_READ | GENERIC_WRITE,
                     0,
```

```
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL);
if (hRegSvc == INVALID_HANDLE_VALUE)
{
      RETAILMSG(1,(TEXT("Unable to open process registration
service.\r\n")));
}
else
{
      // Register a new process
      bRet = DeviceIoControl( hRegSvc,
                              IOCTL_OEM_REGISTER_PROCESS,
                              g_szSampleProc,
                              wcslen(g_szSampleProc),
                              NULL,
                              0,
                              &dwBytes,
                              NULL);
      // (Code here)
      // Deregister the process
      bRet = DeviceIoControl( hRegSvc,
                              IOCTL_OEM_DEREGISTER_PROCESS,
                              g_szSampleProc,
                              wcslen(g_szSampleProc),
                              NULL,
                              0,
                              &dwBytes,
                              NULL);
}
CloseHandle(hRegSvc);
```

Note that applications running from secure SD card must first obtain their actual process name with a call to GetModuleFilename(), as the name will have been randomly generated by the OS. It will be of the form "**sdx0001.tmp**".

The file path returned by this function must be removed before passing the process name to the registration service.

6.1.5.2. Monitoring Power State Transitions

First create a message queue and request power notifications:

```
hMsgQueue = CreateMsgQueue(NULL,&mqoTemp);
hPN = RequestPowerNotifications(hMsgQueue,PBT_TRANSITION);
```

When you receive the notification, compare the system state to the expected state:

```
pPB = (PPOWER_BROADCAST)pBuffer;
if (wcscmp(pPB->SystemPowerState,_T("housekeepingidle")) == 0)
{
      // Pause game, audio playback threads, etc.
}
```

Applications can monitor the transition to any power state in this way, so the transition out of “housekeepingidle” can also be detected. When the unit leaves the housekeeping state, applications and games should un-pause and be ready to continue running as normal.

Revision History

Rev.	Date	Description	Author
0.1	Nov 2003	First Draft for comment	GW /MT
0.2	5 Jan 2004	General changes to all areas.	GW /MT
0.3	20 Oct 2004	Updated for latest drivers	GW /MT
0.4	25 Oct 2004	Added BT enable/disable	GW /MT
0.5	23 Nov 2004	Updated MIDI description to match driver implementation	GW /MT
1.0	8 Dec 2004	Added power management section and general updates	MT
1.1	10 Dec 2004	Updated following Gizmondo Certification Meeting: Added Gizmondo Certification Requirements reference Added backlight and battery driver sections. Expanded piano key handling. Moved security to a main section. Added CHARGERFAULT_MSG Added SWITCHTOSHELL_MSG parameter descriptions	MT
1.2	4 Jan 2005	Document taken over by Gizmondo Studios	PH
1.3	11 Jan 2005	Updated References and Sample Code information	PH
1.4	25 Jan 2005	MIDI driver replaced by MASG driver which supports MP3 as well as MIDI (v3 hardware and above)	MT
1.5	1 June 2005	SD Card Events, SD Copy Protection, SIP Grid documentation, updated Power Management.	PH