MOPHUN™ PROGRAMMING GUIDELINES

Version 1.51

20 October 2003

CONTENTS

INTRODUCTION

This document is targeted to programmers who want to start developing mophun™ applications. It is divided into several parts where the first part describes how to set up a mophun™ project, compile it and execute it in the mophun™ emulator.

The second part of this document describes the different parts of the mophun™ gaming API to help avoid common mistakes.

In the SDK there are several tutorials that use different functionality of the API. The third part of this document describes the tutorials, and how to build and execute them. Feel free to get ideas from them.

The last part has some miscellaneous tips and tricks.

The latest version of this document is available on http://www.mophun.com and for more information please visit the mophun™ developer's forum at the same address.

THE DEVELOPMENT ENVIRONMENT

You can run everything from the command line right out of the box but there are many other ways to set up your development environment for mophun™.

Commercial IDE from MetroWerks

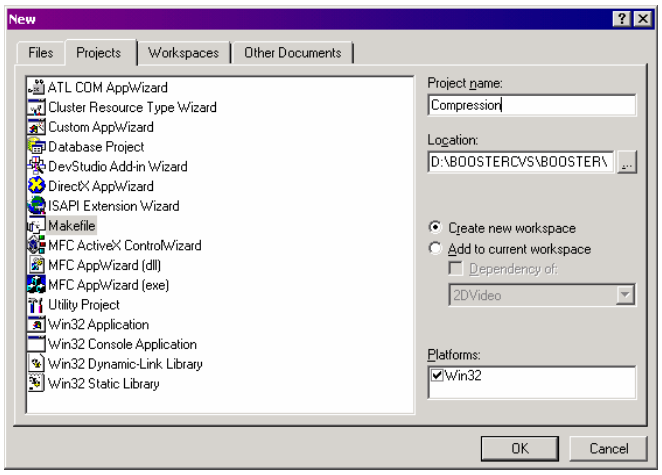
MetroWerks has released mophun™ support in CodeWarrior.

Creating a Microsoft Visual Studio™ 6.0 makefile project

Tip of the day:
Add the compile switch –fvstudio in the makefile to enable double-click on error in Visual Studio. (See the example makefile at step 6)

Here is a description of how to create a project in Microsoft Visual Studio™.

1. Open up Visual Studio™ 6.0
2. Open New on the File menu.
3. Select the Projects tab and find Makefile in the list.
4. Choose a suitable Project name and location and press OK.



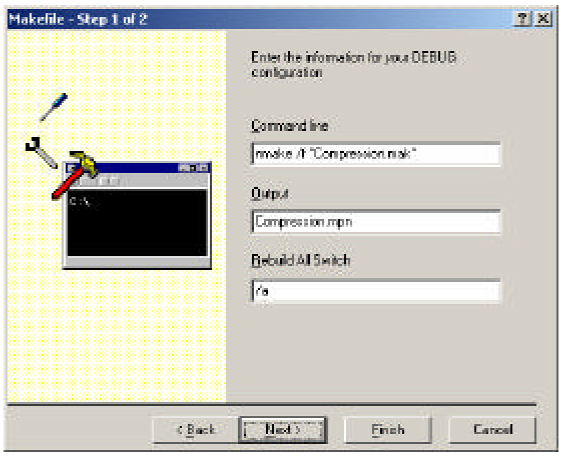
Project name: Compression

Location: D:\BOOSTERCVS\BOOSTER\

(●) Create new workspace
( ) Add to current workspace

Platforms: [✓] Win32

5. Set the .mak file name to the name of your makefile and click Finish.



Enter the information for your DEBUG configuration

Command line: nmake /f "Compression.mak"

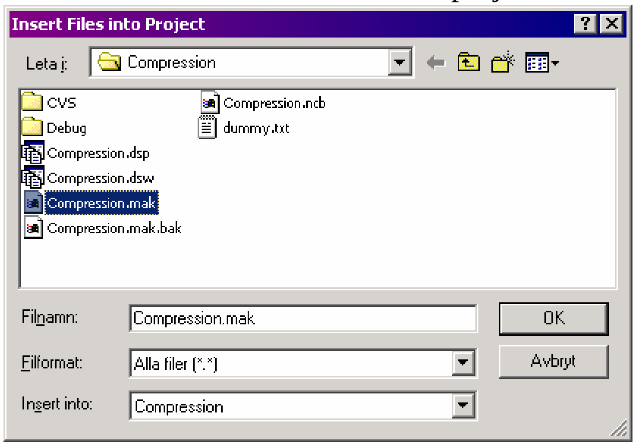
Output: Compression.mpn

Rebuild All Switch: /a

6. Create a makefile. Below is an example of a makefile.

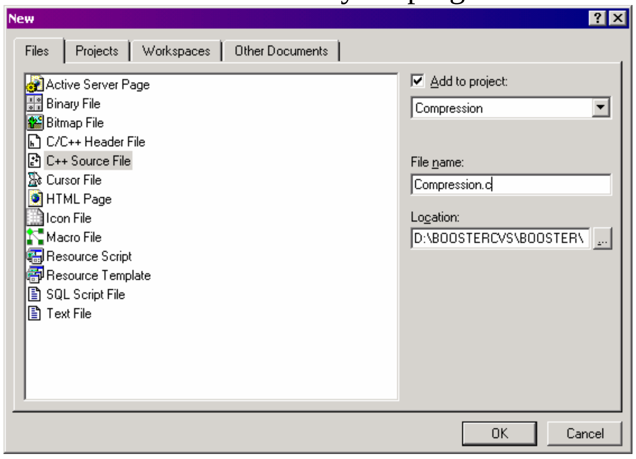
```
# *****************************************
# NAME:    makefile
# PROJECT: mophun™ Tutorials
# DATE:    2002-03-14
# *****************************************

# *****************************************
# FLAGS
#
# Use –g to enable debugging
# Add –fvstudio to enable Double click on error in Visual Studio
# *****************************************
CFLAGS   = -O2

# *****************************************
# OBJECT FILES
# *****************************************
OBJS = Compression.o

all: Compression.mpn

# *****************************************
# COMPILER
# *****************************************
.c.o:
   pip-gcc -c $(CFLAGS) -o $@ $<

# *****************************************
# LINKER
#
# -mstack = 512    - This is the desired STACK size
# -mdata  = 10000  - This is the desired HEAP size. This is the memory
#         used when doing memory allocations.
#
#  Note: remove –s if you want to use the debugger.
# *****************************************
Compression.mpn: $(OBJS)
   pip-gcc -o $@ $(OBJS) -mstack=512 -mdata=10000 -s –ldatacert
```

7. Save the .mak file in the same directory as the project files.
8. Include the makefile to the project. Select Project, Add to Project, Files from the menu:



Leta i: Compression
Filnamn: Compression.mak
Filformat: Alla filer (*.*)
Insert into: Compression

9. Create a .c file for your program. Select File, new on the menu:



☑ Add to project: Compression
File name: Compression.c
Location: D:\BOOSTERCVS\BOOSTER\

10. Name your .c file and press ok. (Compression.c in this example to match the .mak file)
11. Start Coding! Below is a valid mophun™ program.

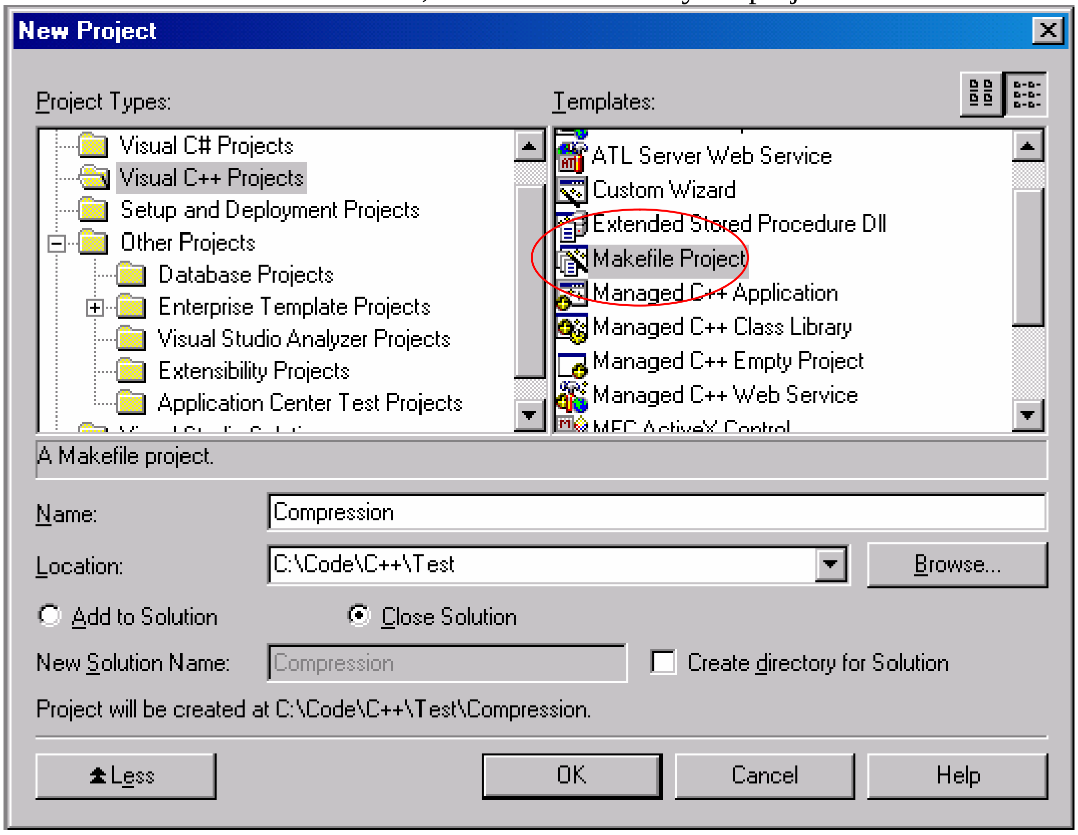
```
#include <vmgp.h>

int main(void)
{
   vClearScreen(vRGB(255,0,0));  // Fill the background with red
   vFlipScreen(1);      // Update the screen

   while(!vGetButtonData());  // Wait for key press

   return 1;
}
```

12. Set the correct[1] path to the mophun™ compiler under Tools – options – Directories (Executable files in the drop down combo). The PATH may also be set in the makefile.
13. Press **F7** to compile.
14. Test the .mpn module in mophun™ emulator.

[1] e.g. *c:\mophun\bin* (depending on where you have installed mophun)

Using Microsoft Visual Studio™ .NET

1. Create a makefile as above
2. Start Visual Studio .NET, choose a name for your project and choose: Makefile Project:



A Makefile project.

Name: Compression

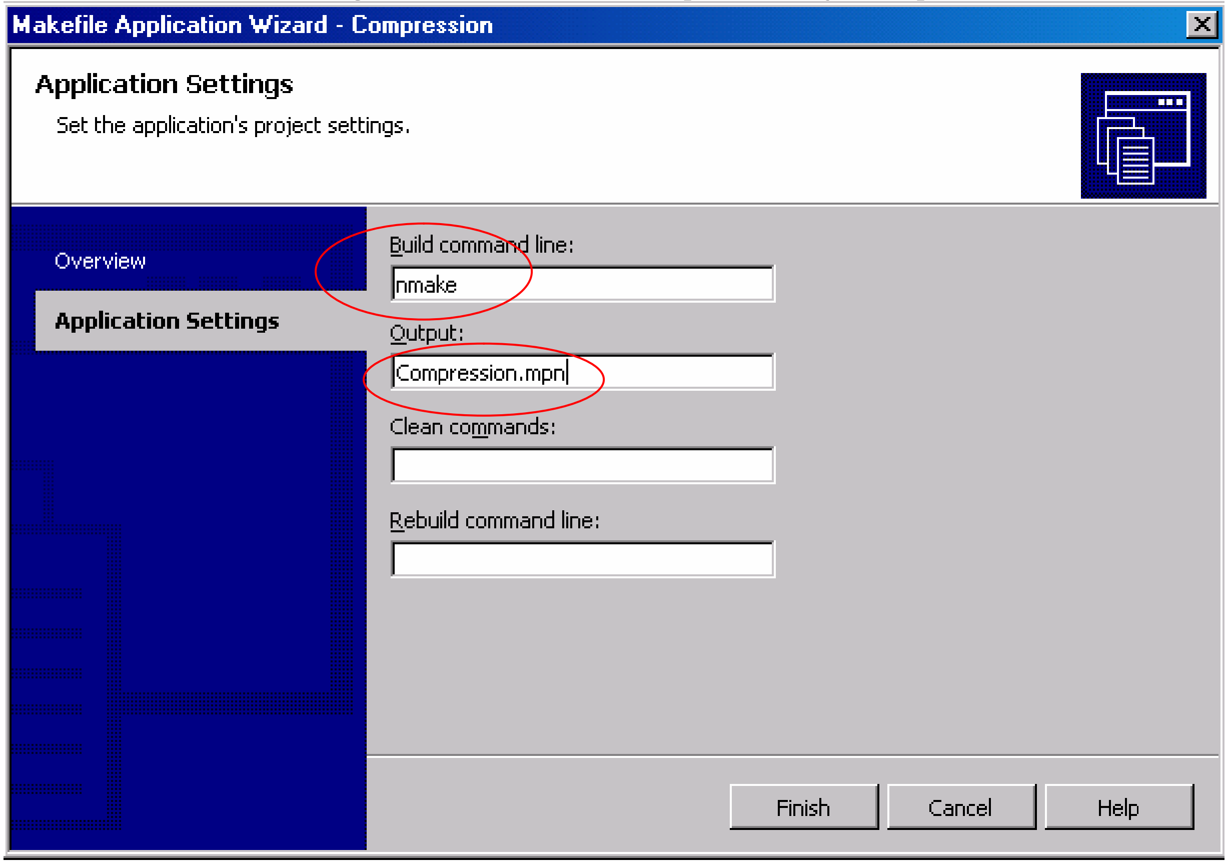
Location: C:\Code\C++\Test

Add to Solution [✓ Close Solution]

New Solution Name: Compression

Project will be created at C:\Code\C++\Test\Compression.

3. After pressing OK, a dialog pops upp, press Application Settings and enter “nmake” in the “Build Command Line”, and change the extension of the output-file to your mpn filename:



Makefile Application Wizard - Compression

Application Settings

Set the application's project settings.

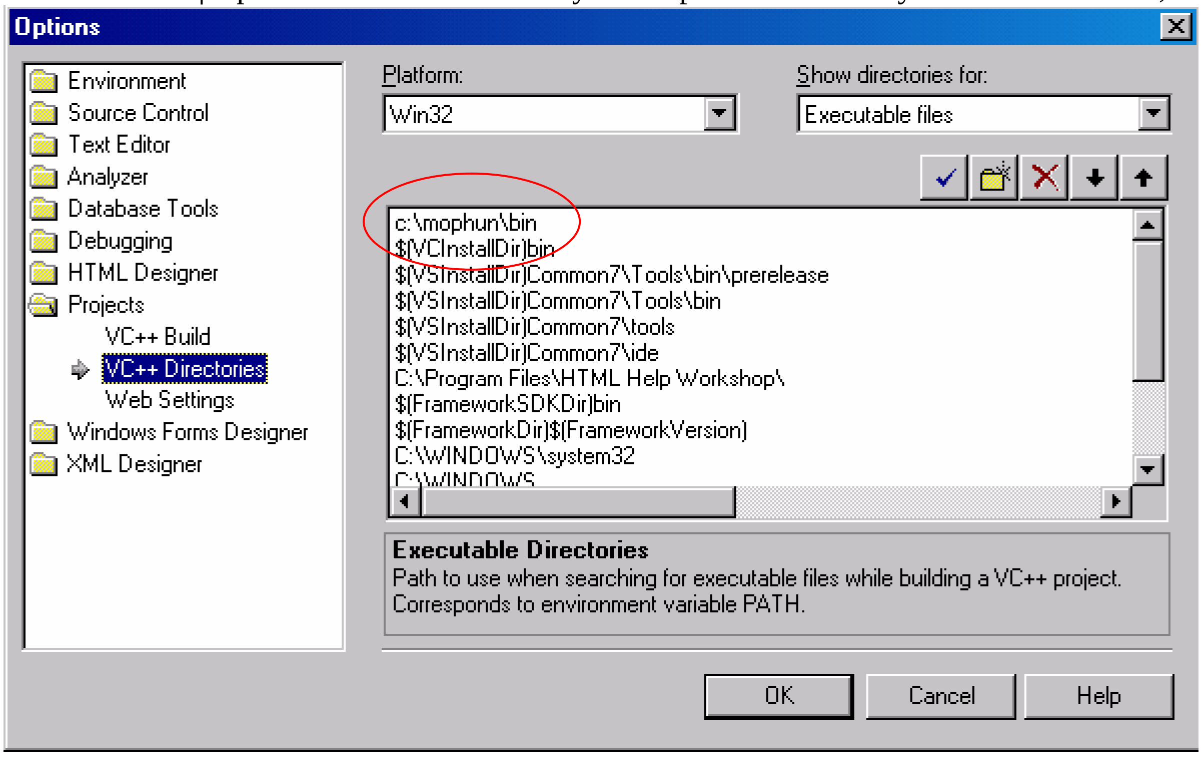
Build command line: nmake

Output: Compression.mpn

Clean commands:

Rebuild command line:

4. Go to Tools | Options and make sure that your mophun/bin directory is in the list. If not, add it:



Options

Platform: Win32

Show directories for: Executable files

```
c:\mophun\bin
$(VCInstallDir)bin
$(VSInstallDir)Common7\Tools\bin\prerelease
$(VSInstallDir)Common7\Tools\bin
$(VSInstallDir)Common7\tools
$(VSInstallDir)Common7\ide
C:\Program Files\HTML Help Workshop\
$(FrameworkSDKDir)bin
$(FrameworkDir)$(FrameworkVersion)
C:\WINDOWS\system32
C:\WINDOWS
```

Executable Directories
Path to use when searching for executable files while building a VC++ project. Corresponds to environment variable PATH.

Note: when compiling an erroneous file, the errors can be found by pressing the “Output” tab.

DEVICE SPECIFICATIONS

To get a better understanding of the target terminal you will be developing for, we advice you to carefully go through the header file “vmgpSonyEricsson.h” found in “/pip/include” directory under your mophun™ install directory.

Also visit www.mophun.com and click on “Phones” to see the details for each device.

FILENAMES

mophun™ filenames can be up to 29 characters. The _ character has a special meaning in mophun™ filenames and should only be used for deployment purposes.

Game files

mophun™ game files always have the extension **.mpn** and contain all the code for a game, and possibly built-in data.

Data files

We recommend that files containing data for extra levels use the **.mpc** extension. This is the extension for data certificates (see page 15).

BUSINESS MODELS FOR MOPHUN™ GAMES

mophun™ games are divided into pay per download games and pay per level games. Any game can also have demo functionality.

Demo functionality

Demo functionality means that the .mpn file (the game itself) can be distributed freely so that the end user can get a feel for the game before buying it. The distributor then uses the VST (Vendor Signing Tool) to signs an empty data ceritificate to unlock the demo. Games with demo functionality can also be purchased directly in which case the VST signs the .mpn file before download, just like games without demo functionality. It’s up to the distributor to choose which approach to use.

For games without demo functionality the VST always signs the .mpn file before download.

Pay per download games

Pay per download games do not have extra levels.

Pay per level games

Pay per level games are games that the user can buy extra levels for. The extra levels always cost money. The data for a level is contained in an .mpc file (data certificate) that is signed by the VST.

Implementation

The game should clearly display its state as early as possible after startup. A straightforward way to indicate when in demo mode is simply to display “DEMO” and to have an in-game menu that lists the available levels. How you choose to do this should be documented in test.txt, see the *Certification Process* document.

API OVERVIEW

This part of the document includes guidelines for the different parts of the API.

Capabilities

Since mophun™ exists on more than one platform, and you want your software to work on as many of them as possible, it is important to check the capabilities of a system instead of assuming that certain features are present.

To show the usage of capabilities, a simple tutorial that prints the system features can be found in the SDK.

Note: Don't forget to set the `size` member of the structure before sending it to vGetCaps:

```
SYSCAPS syscaps;
syscaps.size = sizeof(SYSCAPS);
if(vGetCaps(CAPS_SYSTEM,&syscaps))
{
}
```

Streams & multiplayer

There are several tutorials covering streams in the SDK. They are separated into the following topics:

- **File** - a tutorial that shows how to create and read files.
- **TCP/IP** - a tutorial that uses TCP to connect to a server. The server software (and source) is included.
- **Bluetooth/Infrared** - a tutorial that shows communication using BT or IR. Play the two-dot game.

Note: Don't forget to check the capabilities for supported stream types.

These stream types provide the basic functionality needed to create multiplayer games. TCP/IP (especially over GPRS) or SMS is ideal for turn based games with a game server and BT/IR for local peer-to-peer games where the network latency is not acceptable.

Sound

A tutorial on sound is included in the SDK.

Note: Don't forget to check the capabilities for supported sound types.

2D background mapping library

The background mapping library is used to produce scrolling background images built from 8*8 pixel sized tiles. The backgrounds can contian “auto animation” whereby the graphics may animate on certain tiles without interaction from the program itself. The 2D background mapping library makes it possible to create fast scrolling, multi layered backgrounds even on low end devices.

Described below is a complete mapheader and map including attributes for auto animation, which could be used in a game. The tileset is also included.

Here is some code to create and initialize a map header.

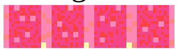
```
mapheader maphead;
maphead.flag = MAP_AUTOANIM | MAP_USERATTRIBUTE;
maphead.format = VCAPS_RGB332;
maphead.width = 12;
maphead.height = 10;
maphead.xpan = 0;
maphead.ypan = 0;
maphead.x = 0;
maphead.y = 0;
maphead.animationspeed = 2;
maphead.mapoffset = MAPADDR;  // address for the map data
maphead.tiledata = TILEADDR;  // address for the tile set
```

This is the actual map and attribute data, the red numbers are attribute information (12*10*2 bytes).

```
001,000,001,000,010,000,001,000,001,000,001,000,001,000,004,000,016,000,005,128,016,000,010,000,
014,000,001,000,001,000,001,000,013,000,001,000,001,000,001,000,016,000,005,128,016,000,012,000,
001,000,001,000,014,000,001,000,001,000,001,000,001,000,014,000,016,000,005,128,016,000,001,000,
012,000,001,000,001,000,001,000,001,000,001,000,001,000,001,000,016,000,005,128,016,000,003,000,
001,000,001,000,001,000,001,000,001,000,001,000,001,000,009,000,002,000,002,000,002,000,002,000,
001,000,001,000,001,000,001,000,001,000,001,000,001,000,010,000,004,000,009,000,002,000,002,000,
001,000,013,000,014,000,001,000,001,000,001,000,001,000,001,000,014,000,004,000,016,000,016,000,
002,000,002,000,002,000,002,000,002,000,015,000,001,000,001,000,001,000,001,000,016,000,016,000,
002,000,002,000,002,000,002,000,002,000,002,000,002,000,002,000,002,000,002,000,002,000,002,000,
016,000,016,000,016,000,016,000,016,000,016,000,016,000,016,000,016,000,016,000,016,000,016,000,
```

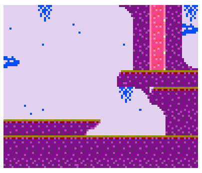
Attribute

The only attribute included is the auto animation attribute (128), but of course all ground tiles could have a user specific “down collision attribute” (e.g 2). 128 = AUTO ANIMATION, the tile will cycle through 4 tiles, see below:



For more information about the attribute format see the **mophun™ API Reference Guide**

This is what the tilemap will look like when it is displayed (it will animate)



Tiles
Here is the tile set for the map above:

Resources

A resource contains some form of data needed by the application. It could be sprites, fonts, sounds, etc. For more complete information read the **mophun™ Resource Compiler Reference**.

The resource data is included by defining it in a resource file for example “res.rc”

Example of a resource file:

```
SPLASH PACKED SPRITE FORMAT RGB332 "splash.bmp"

SECTION DATA
TILES TILESET 8 8 FORMAT RGB332 "tileset.bmp"
FONTA FONT 6 8 FORMAT IND4"font.bmp" "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ <>.!"
```

The above file will include one splash screen that will be compressed, a set of tiles and one font. The res.rc file is passed as indata to the morc (mophun resource compiler) program. morc will produce an object file that can be linked into the game and a header file to be included in the source:

res.h:

```
#define SPLASH 0
#define SPLASH_SIZE  4885
#define SPLASH_OFS   0
#define SPLASH_PACKED
#define SPLASH_UNPACKED_SIZE  7690
#define SPLASH_WIDTH 96
#define SPLASH_HEIGHT   80
#define SPLASH_PALINDEX 0
#define SPLASH_FORMAT   7

extern unsigned char TILES[6400];
#define TILES_COUNT  100
#define TILES_TILESIZE  64
#define TILES_WIDTH  8
#define TILES_HEIGHT 8
#define TILES_FORMAT 7
#define TILES_TILEFMT   7
```

```
#define TILES_SIZE   6400
#define TILES_OFS 0

extern unsigned char FONTA[748];
#define FONTA_CHARCOUNT 41
#define FONTA_CHARSIZE  12
#define FONTA_BPP 2
#define FONTA_WIDTH  6
#define FONTA_HEIGHT 8
#define FONTA_PALINDEX  0
#define FONTA_FORMAT 4
#define FONTA_SIZE   748
#define FONTA_OFS 6400
```

To use the resource data in the application you have to open the resource and read the data.

```
#include “res.h”


SPRITE    *splash;

FONT   FontA;

char   *tileset;

int32_t  myhandle;

/* decompress the splash screen data into a sprite resource structure*/
myhandle= vResOpen(NULL,SPLASH);
splash = vNewPtr(SPLASH_UNPACKED_SIZE);
vDecompress(NULL, splash, myhandle, 1024);
vStreamClose(myhandle);

/* point to the tiles, the pointer could be indata to the map engine */
tileset  = TILES;

/* initialize the font structure */
FontA.width = FONTA_WIDTH;
FontA.height = FONTA_HEIGHT;
FontA.bpp = FONTA_BPP;
FontA.palindex = 0;
FontA.chartbl = FONTA; // character lookup table
FontA.fontdata = FONTA+256; // the bitmap data for the font
```

Data certificates

Introduction

A data certificate is a piece of data that has been digitally signed before being delivered to the end user. Data certificates are used for:

- For the game to verify that it has been paid for. If the data certificate is valid the game unlocks functionality as appropriate.
- To pass information from the distributor with the game or level at the time of purchase. (This is done using "tags".) The information passed can be the identity of the end user which is used to enforce the mophun™ DRM solution and for high-score submittal etc. It can also be game specific information such as a server IP addresses for a multiplayer game for example. This obviously requires the participation of the distributor.

  Even if your game does not require any information there may still be tags inserted by the distributor into the data certificate to handle dates for a subscription based model for example. All tags starting with "M" (as in mophun™) are reserved for this.
- Containing extra levels

Data certificates are created on the server side by the Vendor Signing Tool (VST) during the purchase transaction. The VST does this by adding a header and a digital signature to a data file.

Synergenix will only certify games that use data certificate functionality.

IMEI locking DRM

mophun™ has an optional Digital Rights Management solution that locks a game (or credits/levels etc) to a specific mobile phone on the fly during the purchase transaction using the IMEI (or "UID", a hashed version of the IMEI that can be obtained from the vUID() function).

Data certificate are used in the exact same way regardless if the IMEI locking DRM solution is activated or not. Synergenix patches the game at certification time to enable or disable the IMEI checking.

Types of data certificates

Data certificates appear as embedded within a mophun™ game and as standalone files.

Embedded Data Certificates

Embedded data certificates are used when the game is purchased directly (the game may or may not contain demo mode functionality). Should the game later be forwarded to someone else it will go into demo mode if available, otherwise it should exit.

An embedded data certificate can only be delivered within a game, added to the resource section of the game.

Standalone Data Certificates

Standalone data certificates are used to unlock game with demo functionality and for new levels. Only the data certificate file is locked to a specific terminal.

For demo functionality the VST signs an empty file and delivers it with the filename being the game name but with the .mpc extension.

For new levels the VST signs the data files with the actual level data. The .mpc filename is supplied by the developer, for example `game1_level2_mpc.mpc` (when it arrives on the phone it will be installed as `level2.mpc` in the `game1` directory).

Using data certificates

Checklist

Here is a checklist on what a developer must do to include the data certificate support. See the API reference for documentation on the data certificate functions.

Include placeholders

The resource section of the game has to have a placeholder for the data certificate. Using morc, the resource compiler, add the following at the top of the resource file:

```
#include <datacert.h>
```

This includes the necessary definitions for data certificates. After the METAINFO resource, add the following

```
DATACERT_AND_SECURITY_RESOURCE
```

For games without demo functionality

- ✓ The developer has to add a piece of code at startup that calls on of the data certificate functions. If it returns DATACERT_SUCCESS the game can continue, if it returns DATACERT_FAILURE the game must exit. Otherwise it should display a message that the game has expired and then exit.

For games with demo functionality

- ✓ For games with demo functionality the developer has to add a piece of code that checks for valid embedded and standalone data certificates. The filename of the standalone data certificate to look for is the same as the game filename as it will be on the phone but with the .mpc extension. The code must then call the on of the data certificate functions to check the .mpc files contents. If the function returns DATACERT_SUCCESS for the embedded or the standalone data certificate the game can continue. If both of the certificates return DATACERT_FAILURE the game must go into demo mode. In any other case it should display a message that the game has expired and then exit. (See the tutorials code if this seems complex).

For pay per level games

- ✓ For pay per level games the developer has to add a piece of code for the external extra levels. The game needs to read the data certificate file with the extra level into memory and then call on of the data certificate functions. If it returns DATACERT_SUCCESS the game can go on to read the tags. The data follow immediately after the tags. Remember that the data may need to be aligned before usage if read straight from the file. The game can then use the data to make the new level available to the user.

Document the functionality

- ✓ How the demo functionality and extra levels appear in the game should be documented in the test.txt file (see the *Certification Process* document). Game specific tags should also be documented there

Compiler link switch

- ✓ "-ldatacert" must be included for the declaration of the data certificate functions. *(Example: pip-gcc -o $@ $(OBJS) -mstack=1024 -mdata=2048 –ldatacert)*

Testing data certificates during development

You should verify your data certificate functionality in the emulator after passing your game through the SDK tool `certtest.` The syntax is:

```
certtest [-imei number] [-uid number] [-category number] [-tag tags] infile outfile
```

The emulator has the following imei and uid numbers:

```
imei  =  12345678901234
uid   =  999999999
```

To insert multiple tags either use –tag argument several times or send in all the tags in one argument separated by ':'.

Since certtest takes IMEI/UID as arguments it is possible to check the behavior of your code for both valid and invalid IMEI/UID numbers. You should also test the expiration features using the –tag argument:

- To force your game into demo mode (if you have it) pass in an invalid IMEI number, for example `-imei 12121212121212`
- To force your game to be expired, pass in a subscription period that occurs in the past, for example `–tag M00119990101:M00319990102`
- To make sure that you have reserved enough space in the placeholder for the data certificate pass in all possible tags with their respective max lengths. If your game does not use specific tags this would correspond to the max length of the standard tags only:

  `M00120030101:M00220030108:M00320100131:M01012345678901234567890123456789012`

  If the placeholder is not large enough the data certificate check fails.
- If you have demo functionality the expiration feature should be tested with external data certificates as well.

Use the script in the SDK to test this: `DataCertificateTestSuite.cmd`

The category parameter declares which device category should be used. It is needed because the security system changed between category 2 and 3. The *Certification Process* document contains the list of device models within each category.

Data Certificate Tutorials

The Eel tutorials use data certificates, see page 21.

TUTORIALS

There are several tutorials included in the SDK. They all cover different topics of the API and are described below. In order to be able to compile the tutorials you have to make a change in the makefile for each tutorial. The PATH variable has to be updated so it reflects the path where you installed the mophun™ SDK.

```
# *****************************************
# PATH TO mophun(TM) COMPILER
# *****************************************
PATH = C:\mophun\bin
```

If Microsoft Visual studio is used the path need to be set under Tools – options – Directories (Executable files in the drop down combo).

Capabilities

A simple tutorial that prints the capabilities of the system. Nothing complicated. Just compile it and execute.

Sound

The sound tutorial plays a little tune for you.

Files

The file tutorial creates a file, writes data to it and closes it. Then it reopens it and reads back the data.

TCP/IP

The TCP/IP tutorial consists of two programs. One server that runs on Windows™, and one client that runs on mophun™. Start the server first, and then the client. The client connects to the server, and you will be able to send strings of text to the server by pressing the arrows, or sending strings of text from the server to the client, by typing a string and pressing enter.

Bluetooth / Infrared

The BT/IR tutorial is the simple two-dot game. Start the application on two devices, select method of connection (BT/IR). Then start server on one of the devices and client on the other. When they connect, play the game.

2D Video Library Tutorials

The 2D video tutorials consists of Drawing objects (bouncing balls) that bounce around on the screeen. The graphics used in the tutorials are in 8bit (RGB332) .raw format.

2D Background Mapping Tutorials

The 2D background tutorials draws a scrolling background with the background mapping functionality and draws bouncing sprites on top of it. Setting up sprites, setting up the background engine are shown in this tutorial.
The graphics used in the tutorials are in 8bit (RGB332) .raw format.

AnimAndTransp Tutorial

Demonstrates how to use the automatic animation feature, and per tile transparency of the mapengine.

Sprite Collision

A simple tutorial that demonstrates how to use the useful collision detection for sprites.

SMS tutorial

The SMS tutorial explains how to send and check for SMS in the phones SMS inbox.

Input

A tutorial that shows how to use the input functionality. The program will check the capabilities and run tests for directional keys, the pointer and ASCII input if the input form is supported on the platform.

Task

The task tutorial will create ten tasks that will listen on a channel each and an additional task that will listen for input from the user. When the input task receives a 'KEY_ENTER' command it will send a command 'SHUTDOWNTHREADS' to the main task. When the main task receives the command 'SHUTDOWNTHREADS' it will shut down the input task and send a command 'KILL' to the ten child tasks. When the a child task receives the 'KILL' command it will respond to the main thread, the main thread will wait for the child to respond, when it does, the main thread will dispose the child thread.

Resources

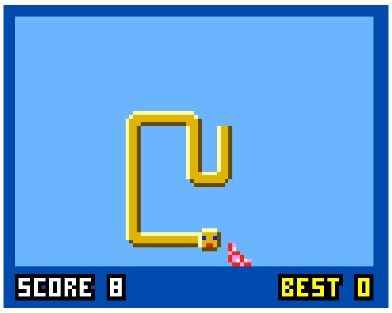
The resource tutorial shows how to use resources in the form of sprites, tileset and raw data. It also shows how to uncompress compressed resource data.

Compression

The compression tutorial shows how to load a compressed bitmap. It allocates memory and decompresses the compressed data to that memory and finally add a sprite header to the beginning of the data.
The sprite will be drawn to the screen in the form of a mophun™ logo. The program will run until a key is pressed.

The Eel

The Eel is a simple game where you control a hungry eel that eats shrimps. The Eel gets longer and longer for every shrimp it eats, and the goal is to eats as many shrimps as possible while avoiding collisions with the edges of the pond or with The Eel itself.



The tutorial explains how to put things together in a larger mophun™ project than the other tutorials. Topics that are covered in this tutorial are:

- capabilities
- fonts
- read data from resource
- save data (highscore) in resource
- input
- timing
- data certificates
- …

Feel free to play around with the code, and create your own quake-killer.

There are different versions of The Eel tutorial:

- TheEel – without any check for data certificates. Can be tested in the emulator directly
- TheEelCertPPD – Pay Per Download version. Checks for internal data certificates
- TheEelCertPPDD – Pay Per Download version with Demo functionality. Checks for internal and external data certificates
- TheEelCertPPL – Pay Per Level version. Checks for internal and external data certificates containing a new level (no demo functionality)

WRITING CROSS PLATFORM APPLICATIONS

With mophun™ this is finally possible, but with different screen sizes and hardware it doesn't come for free and it will not always be possible to create a game that runs well on more than one device but we encourage you to consider if your game could run cross platform. When you submit your game for certification specify if you're targeting:

- a specific device,
- several devices or
- whole *categories* of devices.

The device categories are defined in the *Certification Process* document. If you target whole categories of devices you should test your game with various settings within those categories in the emulator. There are imaginary profiles for various categories in the emulator to get you started.

Developers that want to target multiple platforms need to consider the following:

Display

Always set the desired display window width / height using vSetDisplayWindow. If your choosen display width / height is smaller than that of the actual screen your game will be centered on the screen or streched (highend terminals will use antialiasing). Check video capabilities for actual screen size if you wish to adapt your game to different resolutions for instance showing more of a maze or playfield.

Some devices allow the user to change the screen orientation. You can override the user setting by calling vSetOrientation if your game only supports the default orientation.

Timing

A game that is written and tested for a low end device may run too quickly on a high end device. Therefore, developers need to put in timing code to specify the maximum speed. While this applies to arcade games in particular (your chess computer will never be too fast :-) there is almost always some GUI that may need timing. Try your game on different hardware if available or using different profiles in the emulator. Example:

```
while(vGetTickCount() < dwTickCounter);
dwTickCounter = vGetTickCount() + msperframe;
```

Capabilities

Remember to use vGetCaps function to check for any caps that might affect your application. For example if your game has a peer-to-peer multiplayer feature using Bluetooth it should check if Bluetooth is available and otherwise not display the multiplayer feature. Another good example is checking if midi is supported. The application might use the beeper instead if no midi support is available.

ADDING RESUME FUNCTIONALITY TO A GAME

When a game is terminated by for example an incoming call (on platforms where such an event just ends a game without asking the player…) the games destructor is called. This is where functionality like saving vital game data can take place. This is what the destructor looks like:

```
__attribute__((destructor)) void savestate(void)
{
   if(gInGameFlag==1)
   {
      SaveResumeData();
   }
}
```

SUPPORTING MULTIPLE LANGUAGES

If possible, add support for multiple languages within a single game instead of making several version of it. The game should start in the language that the user has selected on the device but also make it possible to change language in an in-game menu.

To detect the language the user has selected use `vSysGetCulture`[2].

Run the following code the first time the game is started and store the selected language in a meta data variable. The next time the game is started just use the meta data variable.

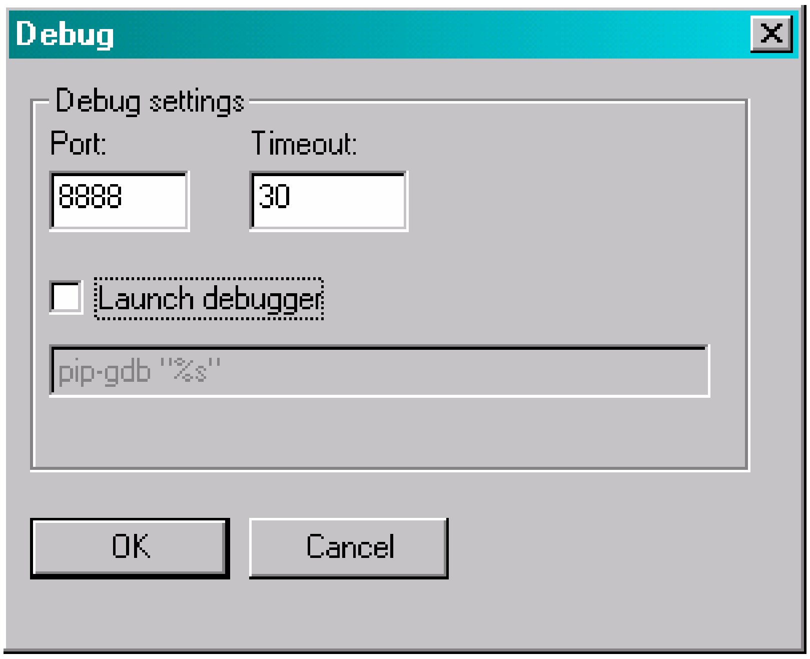
```
culture = vSysGetCulture();
switch (CULTURE_LANGUAGE(culture))
{
   case CULTURE_ZH:
      switch(culture):
         case CULTURE_ZH_CHT:
            // Traditional Chinese
         case CULTURE_ZH_CHS:
            // Simplified Chinese
         ...

   case CULTURE_EN:
      // perhaps not necessary to distinguish between different English cultures?

   case CULTURE_DEFAULT:
      // display the language selection menu

   case default:
      // use default language (English)
}
```

[2] This feature was introduced with mophun RTE 2.0 but is backward compatible: earlier devices will return CULTURE_DEFAULT

DEBUGGING

The debugger used by the mophun™ emulator is a version of GDB with a GUI called insight. The debugger can be launched by the mophun™ emulator, or started separately and then remote connected to the mophun™ emulator.

The debugger has to be configured in order to find the mophun™ emulator. This is done from the "File" menu, "Target settings..." in the debugger. Below are two screenshots from the mophun™ emulator and the debugger with recommended values entered.



Debug

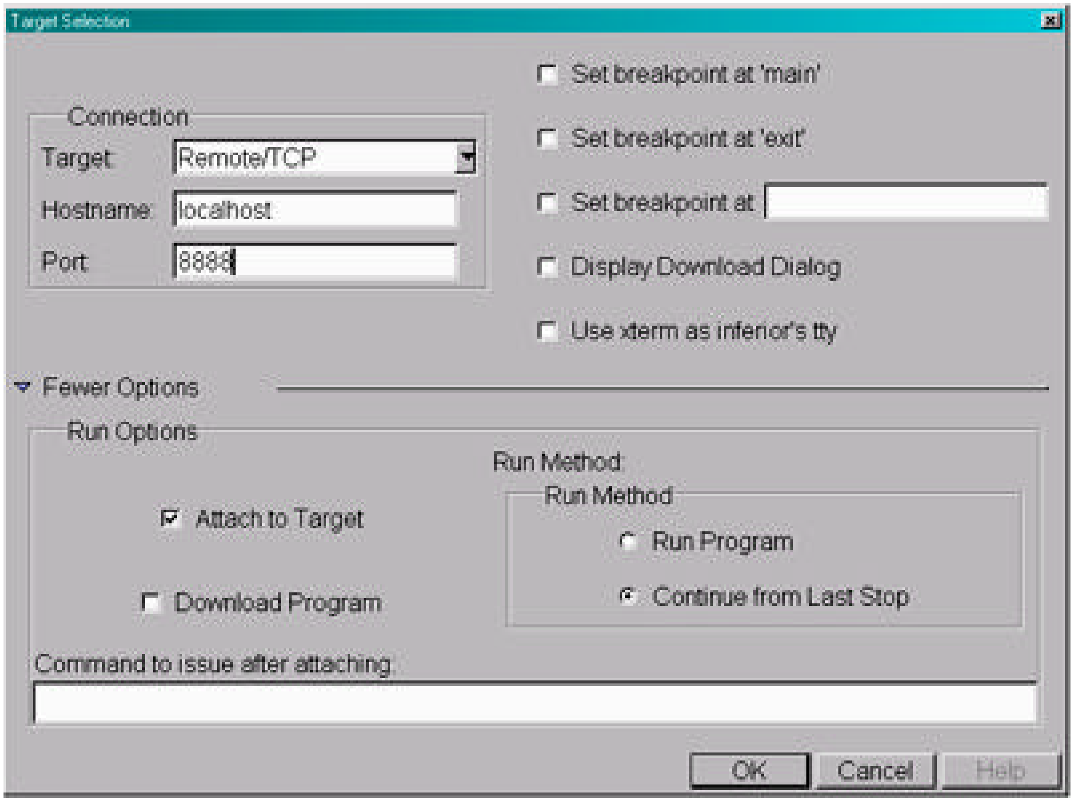
Debug settings

Port: 8888

Timeout: 30

☐ Launch debugger

pip-gdb "%s"



Target Selection

Connection

Target: Remote/TCP

Hostname: localhost

Port: 8888

☐ Set breakpoint at 'main'

☐ Set breakpoint at 'exit'

☐ Set breakpoint at

☐ Display Download Dialog

☐ Use xterm as inferior's tty

Fewer Options

Run Options

☑ Attach to Target

☐ Download Program

Run Method

Run Method

○ Run Program

● Continue from Last Stop

Command to issue after attaching:

The debugger can be launched without the GDI by running the command “pip-gdb –nw”. To connect to the mophun™ emulator from the console debugger type the command "target remote localhost:PORT".

For extended help with running the GDB debugger type “help” in the console or select the “Help Topics” from the Help menu when running GDB in GDI mode. The GDB manual is also available in the SDK document folder.

APPENDIX A. REVISION HISTORY

Revision	**Date**	**Author**	**Change**
1.48	2003-09-25	Björn Wennerström	Added revision history Added multi language info Added reference to mophun.com | phones Moved emulator info to emulator help file
1.50	2003-10-15	Björn Wennerström	Added info on setting screen orientation, category parameter to CertTest
1.51	2003-10-20	Anders Norlander	Bring data certificate chapter up-to-date with the library.