MEDIATEK

MRE Development Quick Start Guide

Version:	1.0
Release Date:	2011-09-13

Table of Contents

1 Preface

1.1 Purpose

This guide provides a brief introduction to MRE development and how to create a Hello World program with the MRE SDK. .

1.2 Scope

This guide features MRE1.0

1.3 Terminology and Abbreviations

MRE	MAUI Runtime Environment, MAUI is a feature phone software package from MediaTek, and MRE is a runtime environment based on MAUI.
MRE SDK	MRE Software Development Kit
MRE IDE	MRE Integration Development Environment

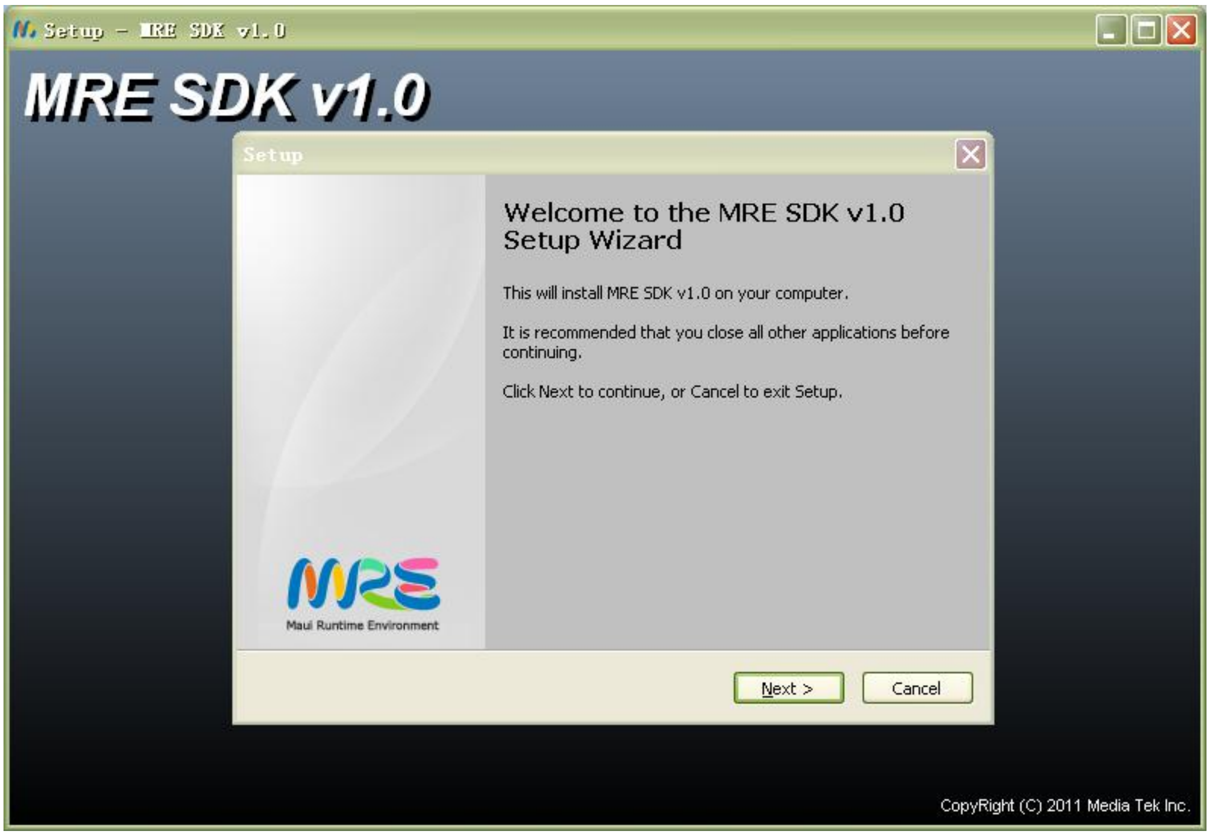
2 Installing MRE1.0 SDK

2.1 Installation Environment

- PC Operating Systems
  - Windows XP
  - Windows Vista
  - Windows 7
- C/C++ Development Tools
  - Microsoft Visual Studio 2008 Express Editor
  - Microsoft Visual Studio 2008 Professional Editor
- ARM Compiler
  - ARM Developer Suite V1.2（ADS1.2）

2.2 Installation Procedures

1. Double-click the installation package MRE_SDK_v1.0.exe and the installation process will begin.
2. Enter Welcome Page
   - Click on the "Cancel" button to terminate the installation process.
   - Click on the "Next" button to continue with the installation process.



Welcome to the MRE SDK v1.0 Setup Wizard

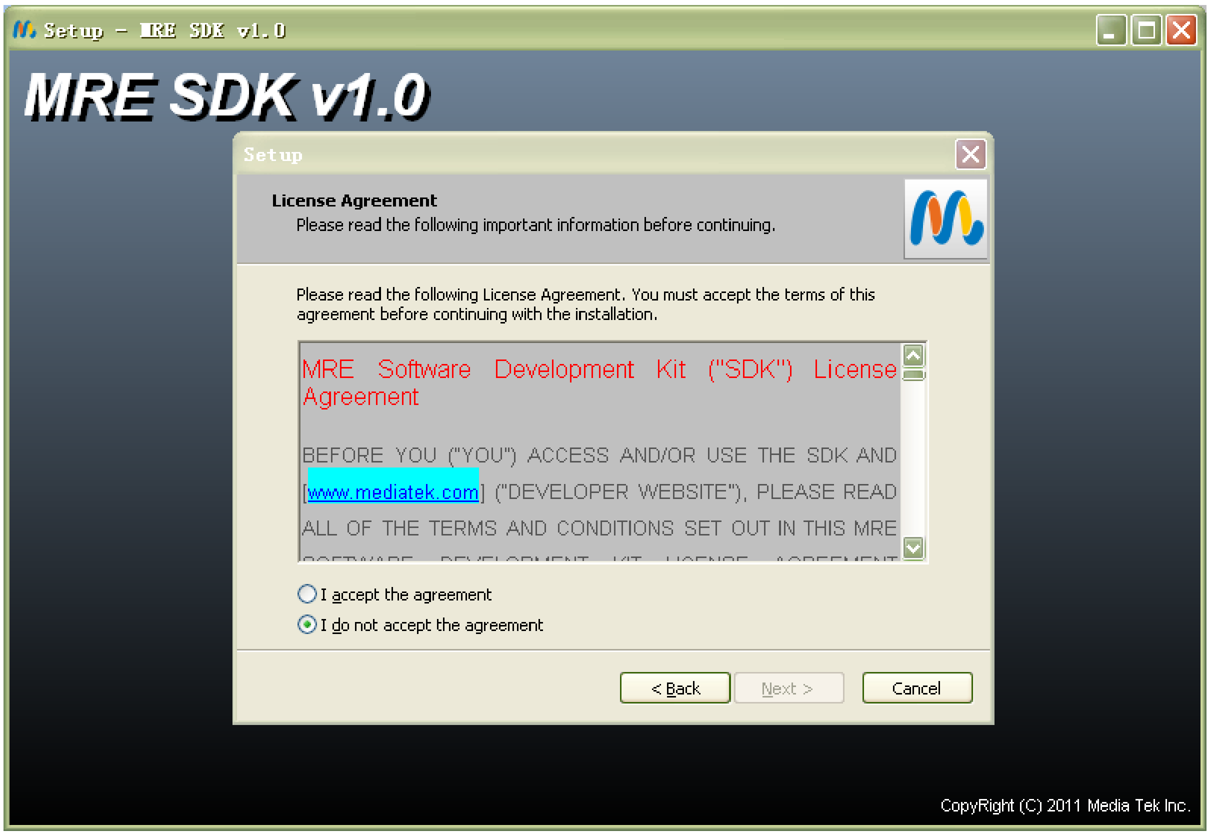
This will install MRE SDK v1.0 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure 2-1. Welcome

3. Enter the License Agreement page; please read the content of the license carefully and click "I accept the agreement" if you agree to the terms and restrictions of this license agreement. The installation process will then continue. If you do not agree to the license agreement, the installation procedure will not proceed to the next stage.



License Agreement

Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

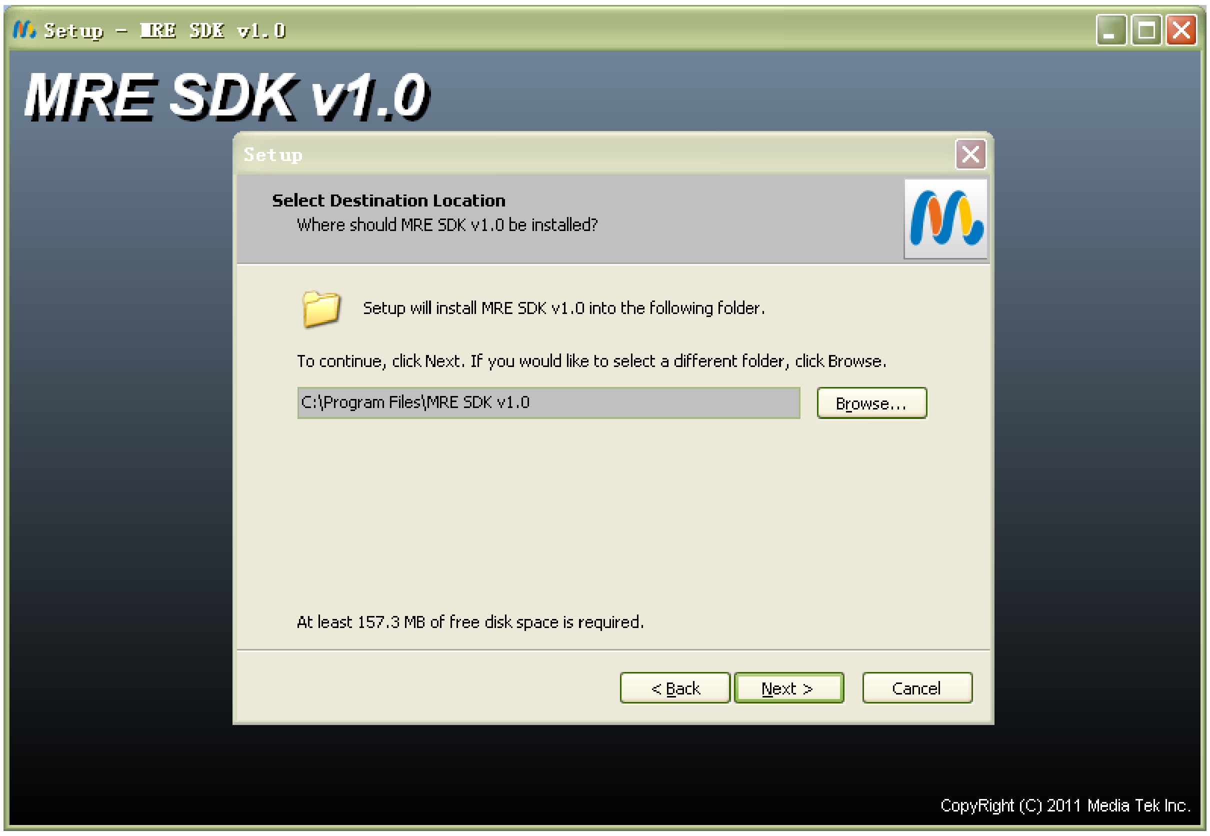
MRE Software Development Kit ("SDK") License Agreement

BEFORE YOU ("YOU") ACCESS AND/OR USE THE SDK AND [www.mediatek.com] ("DEVELOPER WEBSITE"), PLEASE READ ALL OF THE TERMS AND CONDITIONS SET OUT IN THIS MRE

[ ] I accept the agreement
[✓] I do not accept the agreement

Figure 2-2. License Agreement

4. Select the installation path into the page, the default path is "C:\Program Files\ MRE SDK v1.0".User may change the installation folder. Click the "Browse" button and select a new installation location.


Setup - MRE SDK v1.0

MRE SDK v1.0

Setup

Select Destination Location

Where should MRE SDK v1.0 be installed?

Setup will install MRE SDK v1.0 into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

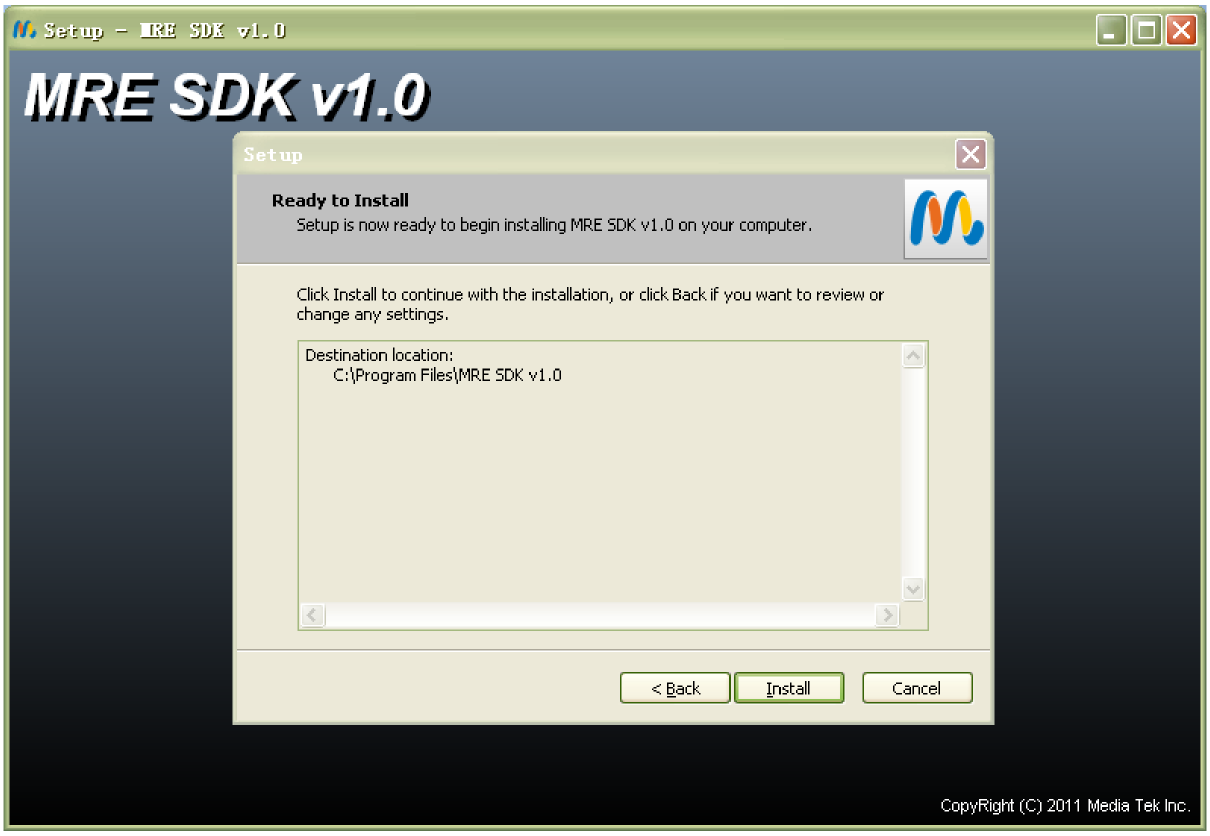
C:\Program Files\MRE SDK v1.0

At least 157.3 MB of free disk space is required.

CopyRight (C) 2011 Media Tek Inc.

Figure 2-3. Select Location

5. Enter the confirmation page; final selected installation folder will be displayed. If you need to change the installation folder, click on the "Back" button to return to the previous page and select a different folder. If you would like to cancel the installation process, click on the "Cancel" button. After you have confirmed that all information is correct, click on "Install" to begin the automated installation process.



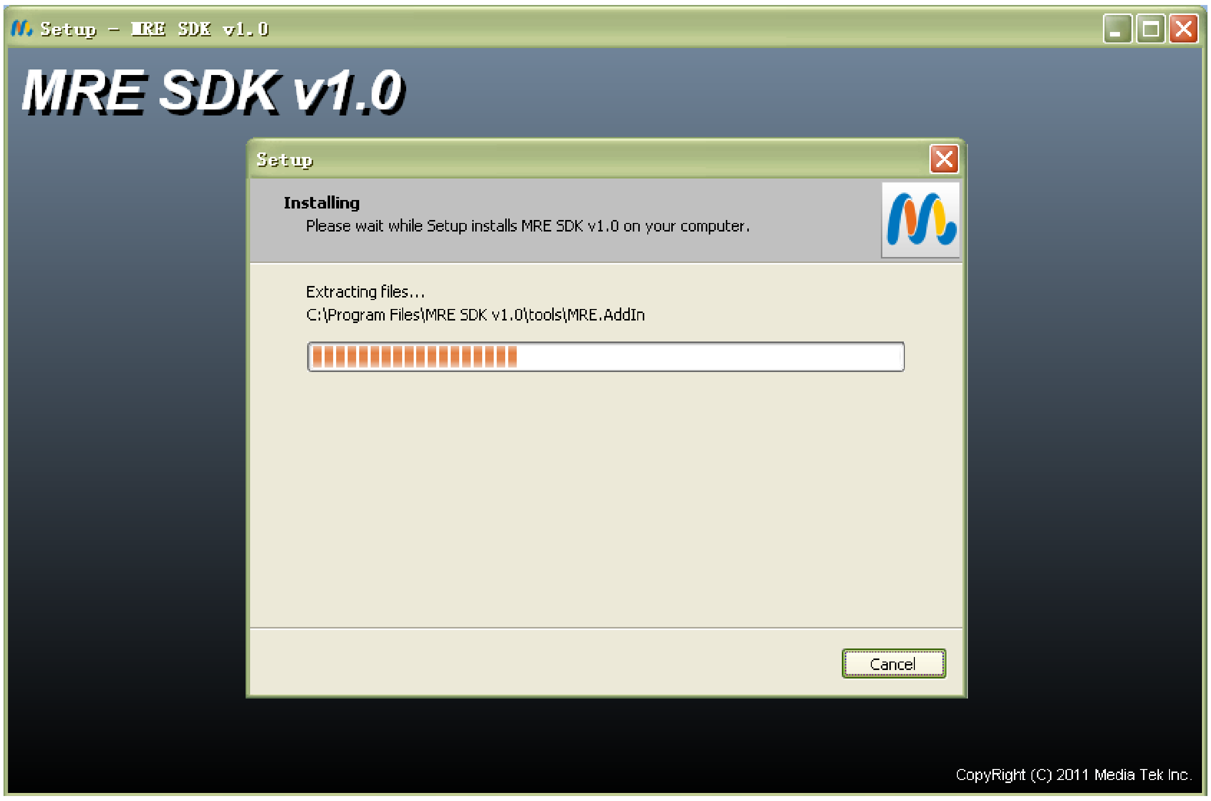
Ready to Install
Setup is now ready to begin installing MRE SDK v1.0 on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

Destination location:
C:\Program Files\MRE SDK v1.0

Figure 2-4. License Agreement

6. Installation Process page. Please wait for the installation to complete, or cancel the installation in progress.



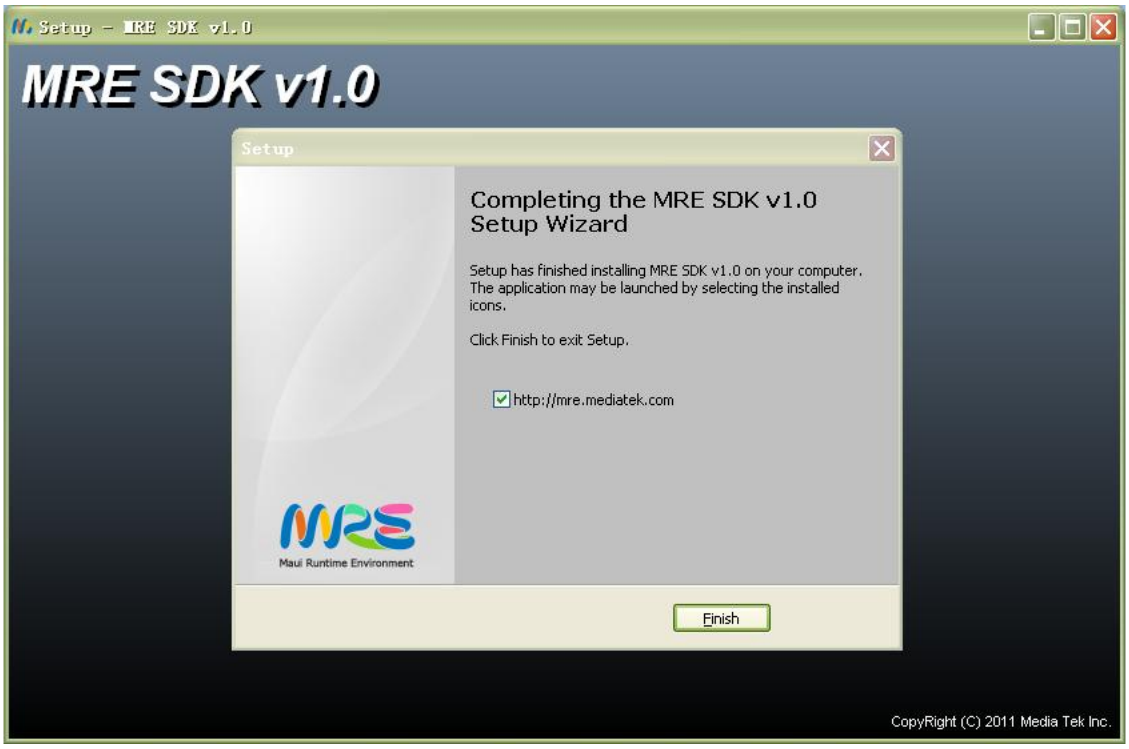
Installing
Please wait while Setup installs MRE SDK v1.0 on your computer.

Extracting files...
C:\Program Files\MRE SDK v1.0\tools\MRE.AddIn

Figure 2-5. Installing Process

7. After installation has been completed successfully, the system will enter the Finished Page. You can visit the MRE SDK website via the link provided on the page to access information

related to MRE.



Completing the MRE SDK v1.0 Setup Wizard

Setup has finished installing MRE SDK v1.0 on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

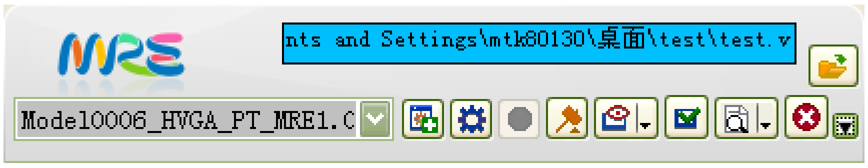
☑ http://mre.mediatek.com

Figure 2-6. Finished Page

3 Creating an MRE Application

3.1 MRE App Wizard

After MRE SDK1.0 has been successfully installed, the express options for MRE SDK IDE1.0 launcher can be seen on Start Menu → All Programs → MRE SDK 1.0.



The main purpose of MRE SDK 1.0 is to provide the user with a platform to develop MRE applications; it includes the following functions:

1. Simulator type selection
2. MRE App Wizard
3. Configuration
4. Generation
5. Other tools
6. Options
7. Help

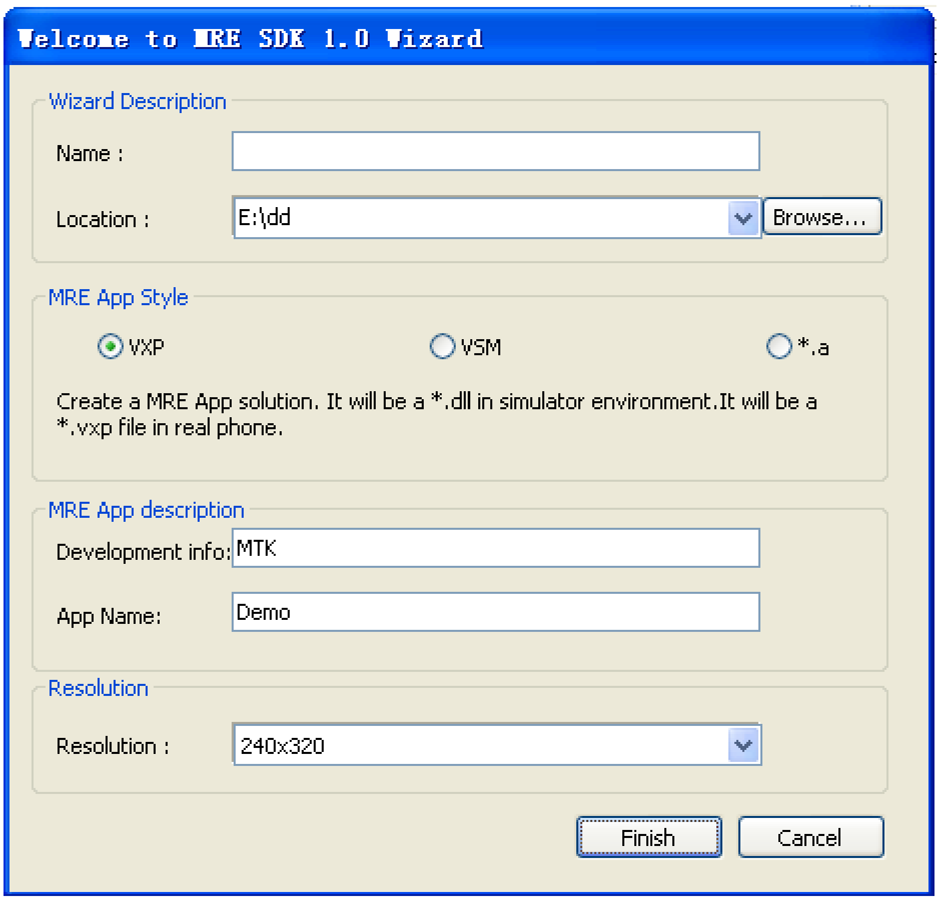
3.1.1 Selecting simulator type

After launching MRE SDK 1.0, the user can select the desired simulator type with the leftmost Model combo box.

3.1.2 Startup Wizard

Launch MRE SDK 1.0 and click to start the MRE Create Application Wizard.
On the Wizard page, add MRE application information that you need to configure.

1. MRE application project name
2. MRE application project storage path
3. MRE application name and developer information
4. MRE application supported resolution (width x height)
5. VXP MRE
6. VSM MRE dynamically loaded runtime library
7. .a MRE statically loaded runtime library



Welcome to MRE SDK 1.0 Wizard

Wizard Description

Name :

Location : E:\dd

MRE App Style

[✓ VXP] VSM *.a

Create a MRE App solution. It will be a *.dll in simulator environment.It will be a *.vxp file in real phone.

MRE App description

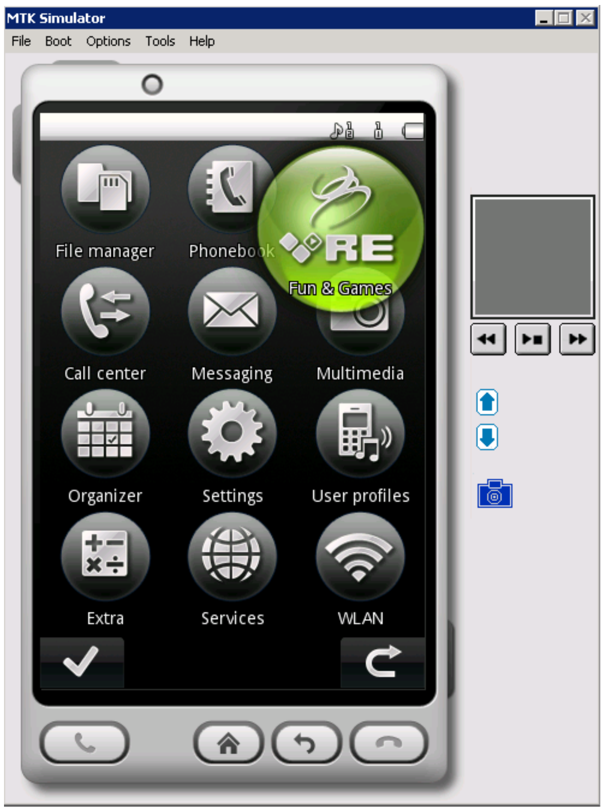
Development info: MTK

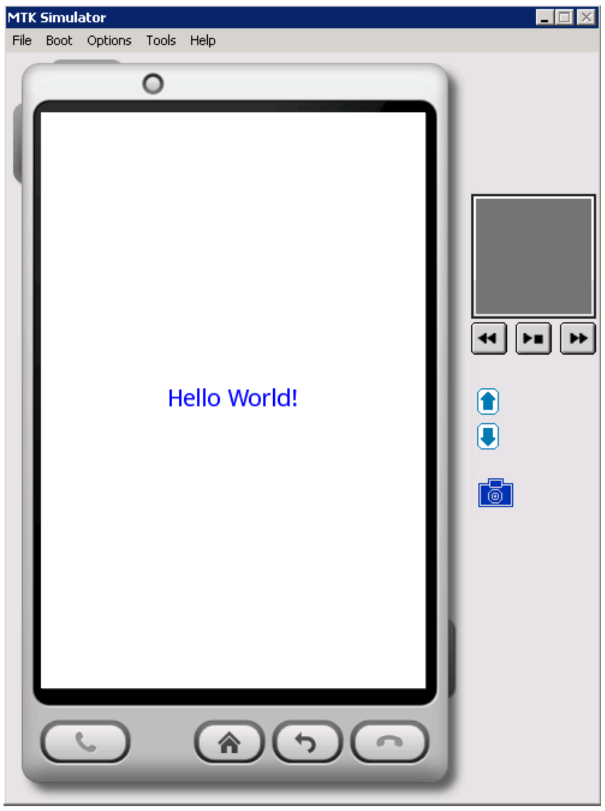
App Name: Demo

Resolution

Resolution : 240x320

The Wizard will help user to generate a simple MRE application project: ’Hello World’.User will need Visual Studio 2008 Professional Editor or Visual Studio 2008 Express Editor to open the generated MRE application project. Use Visual Studio 2008 for compiling and debugging. User can use a simulated handset to invoke the MRE application with MoDIS emulation.





3.2 Hello World

The code for the Hello World application automatically generated by the MRE App Wizard is as follows:

```
#include "vmsys.h"
#include "vmio.h"
#include "vmgraph.h"
#include "vmchset.h"
#include "vmstdlib.h"
/* ---------------------------------------------------------------------------
* Global data.
* ------------------------------------------------------------------------- */
VMINT layer_hdl[1]; //Layer handle array. MRE supports two layers.

/* ---------------------------------------------------------------------------
* Built-in function declaration.
* ------------------------------------------------------------------------- */
/*
* Handling system event.
*/
void handle_sysevt(VMINT message, VMINT param);

/*
* Handling key event.
*/
void handle_keyevt(VMINT event, VMINT keycode);
```

```
/*
* Handling touch event.
*/
void handle_penevt(VMINT event, VMINT x, VMINT y);

/*
* Example.
*/
static void draw_hello(void);

/**
* Application entry function.
*/
void vm_main(void) {
        layer_hdl[0] = -1;
        vm_reg_sysevt_callback(handle_sysevt);
        vm_reg_keypad_callback(handle_keyevt);
        vm_reg_pen_callback(handle_penevt);
}

void handle_sysevt(VMINT message, VMINT param) {
#ifdef          SUPPORT_BG
        /* The application updates the screen when receiving the message VM_MSG_PAINT
        *  what is sent after the application is activated. The application can skip
        *  the process on screen when the VM_MSG_ACTIVE or VM_MSG_INACTIVE is received.
        */
        switch (message) {
                case VM_MSG_CREATE:
                        /* the GDI operation is not recommended as the response of the message*/
                        break;
                case VM_MSG_PAINT:
                        /* cerate base layer that has same size as the screen*/
                        layer_hdl[0] = vm_graphic_create_layer(0, 0,
                                vm_graphic_get_screen_width(),
                                vm_graphic_get_screen_height(), -1);

                        /* set clip area */
                        vm_graphic_set_clip(0, 0,
                                vm_graphic_get_screen_width(),
                                vm_graphic_get_screen_height());

                        draw_hello();
                        break;
                case VM_MSG_HIDE:
                case VM_MSG_QUIT:
                        if( layer_hdl[0] != -1 )
                        {
                                vm_graphic_delete_layer(layer_hdl[0]);
                                layer_hdl[0] = -1;
                        }

                        break;
        }
#else
        switch (message) {
                case VM_MSG_CREATE:
```

```
                case VM_MSG_ACTIVE:
                        /*cerate base layer that has same size as the screen*/
                        layer_hdl[0] = vm_graphic_create_layer(0, 0,
                                vm_graphic_get_screen_width(),
                                vm_graphic_get_screen_height(), -1);

                        /* set clip area*/
                        vm_graphic_set_clip(0, 0,
                                vm_graphic_get_screen_width(),
                                vm_graphic_get_screen_height());
                        break;

                case VM_MSG_PAINT:
                        draw_hello();
                        break;

                case VM_MSG_INACTIVE:
                case VM_MSG_QUIT:
                        if( layer_hdl[0] != -1 )
                                vm_graphic_delete_layer(layer_hdl[0]);

                        break;
        }
#endif
}

void handle_keyevt(VMINT event, VMINT keycode) {
/* exit application if any key event happens.*/
        if( layer_hdl[0] != -1 )
                vm_graphic_delete_layer(layer_hdl[0]);
        vm_exit_app();
}

void handle_penevt(VMINT event, VMINT x, VMINT y){
/* exit application if any pen event happens*/
        if( layer_hdl[0] != -1 )
                vm_graphic_delete_layer(layer_hdl[0]);
        vm_exit_app();
}

static void draw_hello(void) {
        VMWCHAR s[50];
        VMUINT8* buf;
        int x, y, w;

        /* The encoding format of MRE is UCS2 by default. Therefore the text string to be displayed must be transferred to
UCS2 encoding format.
Note: VC++6.0 adopt GB2312 as encoding format by default. Strings can be converted from GB2312 to UCS2 encoding
format by the prefix “L” in VC++6.0 editor. Because ADS1.2 does not support Chinese character, MRE text string
conversion function should be used to implement encoding format conversion if needed.
        */
        vm_gb2312_to_ucs2(s, 50, "Hello, world!");
        w = vm_graphic_get_string_width(s);
        x = (vm_graphic_get_screen_width() - w) / 2;
        y = (vm_graphic_get_screen_height() - vm_graphic_get_character_height()) / 2;

        /* Obtain layer buffer pointer*/
        buf = vm_graphic_get_layer_buffer(layer_hdl[0]);

        /* Clear screen with white color and black background*/
        vm_graphic_fill_rect(buf, 0, 0, vm_graphic_get_screen_width(),
            vm_graphic_get_screen_height(), VM_COLOR_WHITE, VM_COLOR_BLACK);

        /* Output the word to buffer */
        vm_graphic_textout(buf, x, y, s, wstrlen(s), VM_COLOR_BLUE);

        /* Flush the data from buffer area to the screen.*/
        vm_graphic_flush_layer(layer_hdl, 1);
}
```

3.2.1 MRE SDK API

When using the MRE SDK API, it is necessary to include the relevant header files; for more detailed description of the MRE SDK API, refer to the API documentation.

Whether certain aspects of the API can operate correctly depends on the features available on the handset being used.

3.2.2 MRE Application Code Structure

The main entry point for an MRE application is ***vm_main()***.

```
void vm_main(void) {
        layer_hdl[0] = -1;
        vm_reg_sysevt_callback(handle_sysevt);
        vm_reg_keypad_callback(handle_keyevt);
        vm_reg_pen_callback(handle_penevt);
}
```

Usually we register three callback functions in ***vm_main()*** to handle the following three types of messages:

System event messages
Key event messages
Pen event messages

In addition, a few other initialization tasks can also be carried out within this function.

3.2.2.1 System event messages

In the following we introduce the system messages of MRE. At the time of building the application, if the option “support running in the background,” is selected, then it indicates that the application supports background operations, and the program will receive the “**VM_MSG_PAINT**” and “**VM_MSG_HIDE**” messages when entering the foreground and background, respectively. .

VM_MSG_CREATE
When MRE loads an application, this is the first message sent to the application upon return from vm_main() to indicate that the application has been created

VM_MSG_PAINT
When the application window is active, MRE will send this message to the application; all graphics operations can be performed only after this message has been received

VM_MSG_HIDE
For an application that supports running in the background, when the application window is obscured or has been put in background operation, then MRE will send this message to the application

VM_MSG_QUIT
MRE will send this message to the application when it is about to terminate the application

If the application does not support running in the background, when user presses the END or HOME key to return to the main interface, the application will exit. When the application window is obscured by another window, or when it returns, it will receive the "**VM_MSG_INACTIVE**" and "**VM_MSG_ACTIVE**" message, respectively.

VM_MSG_INACTIVE
For an application that does not support running in the background, when the application window is obscured, MRE will send this message to the application; for example, when there is an incoming call, or a new SMS is received, or a USB device is plugged in or unplugged, the application will have to halt all drawing operations

VM_MSG_ACTIVE
For an application that does not support running in the background, when the application window becomes the active window, MRE will send this message to the application. The application will need to continue its drawing operations

If the application is designated to support running in the background at the time it is created, then the compiler option **SUPPORT_BG** will be defined

```
void handle_sysevt(VMINT message, VMINT param) {
#ifdef      SUPPORT_BG
    /* The application updates the screen when receiving the message VM_MSG_PAINT
    *  what is sent after the application is activated. The application can skip
    *  the process on screen when the VM_MSG_ACTIVE or VM_MSG_INACTIVE is received.
    */
    switch (message) {
        case VM_MSG_CREATE:
            /* the GDI operation is not recommended as the response of the message*/
            break;
        case VM_MSG_PAINT:
            /* cerate base layer that has same size as the screen*/
            layer_hdl[0] = vm_graphic_create_layer(0, 0,
                vm_graphic_get_screen_width(),
                vm_graphic_get_screen_height(), -1);

            /* set clip area */
            vm_graphic_set_clip(0, 0,
                vm_graphic_get_screen_width(),
                vm_graphic_get_screen_height());
```

```
                                draw_hello();
                                break;
                    case VM_MSG_HIDE:
                    case VM_MSG_QUIT:
                                if( layer_hdl[0] != -1 )
                                {
                                            vm_graphic_delete_layer(layer_hdl[0]);
                                            layer_hdl[0] = -1;
                                }

                                break;
        }
#else
        switch (message) {
                    case VM_MSG_CREATE:
                    case VM_MSG_ACTIVE:
                                /*cerate base layer that has same size as the screen*/
                                layer_hdl[0] = vm_graphic_create_layer(0, 0,
                                            vm_graphic_get_screen_width(),
                                            vm_graphic_get_screen_height(), -1);

                                /* set clip area*/
                                vm_graphic_set_clip(0, 0,
                                            vm_graphic_get_screen_width(),
                                            vm_graphic_get_screen_height());
                                break;

                    case VM_MSG_PAINT:
                                draw_hello();
                                break;

                    case VM_MSG_INACTIVE:
                    case VM_MSG_QUIT:
                                if( layer_hdl[0] != -1 )
                                            vm_graphic_delete_layer(layer_hdl[0]);

                                break;
        }
#endif
}
```

3.2.2.2 Key event messages

MRE supports the following key event messages:

VM_KEY_EVENT_UP
VM_KEY_EVENT_DOWN
VM_KEY_EVENT_LONG_PRESS
VM_KEY_EVENT_REPEAT

The key code can be obtained from the parameters of the input message. In the Hello World application, the response to any key event messages is exiting the application

```
void handle_keyevt(VMINT event, VMINT keycode) {
```

```
/* exit application if any key event happens.*/
        if( layer_hdl[0] != -1 )
                  vm_graphic_delete_layer(layer_hdl[0]);
        vm_exit_app();
}
```

3.2.2.3 Pen event messages

MRE supports the following pen event messages:

VM_PEN_EVENT_TAP
VM_PEN_EVENT_RELEASE
VM_PEN_EVENT_MOVE
VM_PEN_EVENT_LONG_TAP
VM_PEN_EVENT_DOUBLE_CLICK
VM_PEN_EVENT_REPEAT

The coordinates of a pen event can be obtained from the parameters of the input message. In the Hello World application, the response to any pen event messages is exiting the application

```
void handle_penevt(VMINT event, VMINT x, VMINT y){
/* exit application if any pen event happens*/
        if( layer_hdl[0] != -1 )
                  vm_graphic_delete_layer(layer_hdl[0]);
        vm_exit_app();
}
```

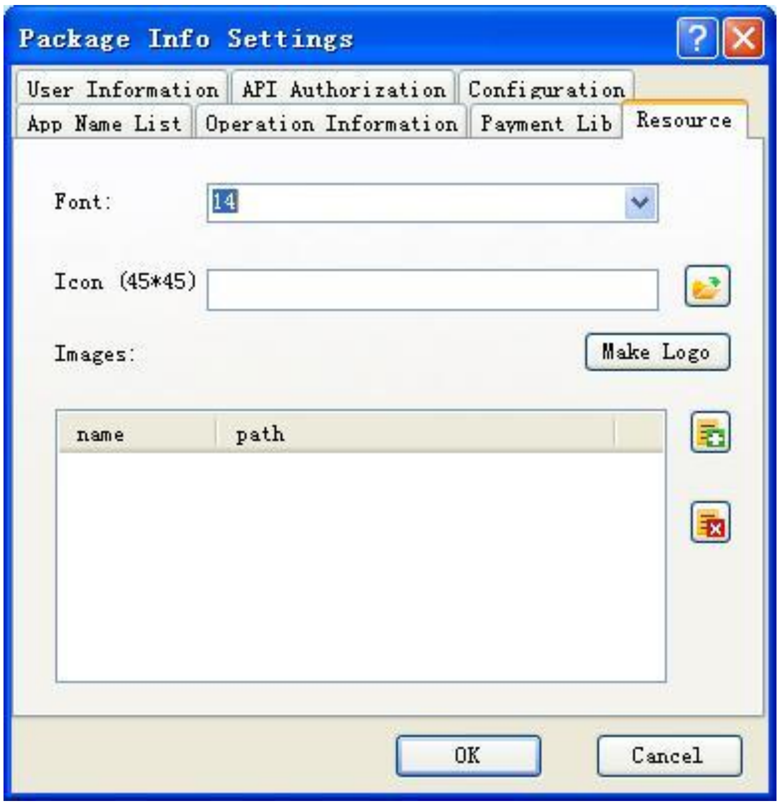
3.2.3 Using Resources

When running, an MRE application is able to access external resources based on its own needs. These external resources are generated by the resource editor and incorporated into the application at the time the application is built. Resources can be of any type, such as common images, audio, video or other binary proprietary formats custom-defined by other applications.

3.2.3.1 Adding Resources

On the Resource Configuration page, one can use to add a resource to the list of MRE application resources Use to remove the selected resources.
When a resource is added, two pieces of information will be generated

- Resource name: In an MRE application, required resource data can be loaded through the resource call interface.
- Resource path: Add resources to a location on the PC.



Font: 14

Icon (45*45)

Images:

3.2.3.2 Load resources in the application

An application can call the API *vm_load_resource* in *vmres.h* to load resources added in the resource editor. The application will need to free the memory in the stored resource data, as shown in the following sample code:

```
VMUINT8 *res_data = NULL;
VMUINT8 *img = NULL;
VMINT res_size = 0;
VMINT hcanvas = VM_GDI_SUCCEED;
struct frame_prop * frame_ptr = NULL;

//MTK_LOGO.jpg is the file name of uploaded resource.
res_data = vm_load_resource("MTK_LOGO.jpg", &res_size);
if (res_data == NULL)
{
        // If failed to load resource, release system resource, and quit the application.
        vm_exit_app();
        return;
}
else
{
        // Put the resource date into MRE graphic canvas.
   hCavas = vm_graphic_load_image(res_data, res_size);
```

```
        // Release the resources after usage.
        vm_free(res_data);

    if (hcanvas >= VM_GDI_SUCCEED){
        frame_ptr = vm_graphic_get_img_property(hcanvas, 1);
        vm_graphic_blt(buf, 0, 0,
            vm_graphic_get_canvas_buffer(hcanvas), 0, 0,
            frame_ptr->width, frame_ptr->height, 1);
    }
}

// Flush the data from buffer area to the screen.
vm_graphic_flush_layer(layer_hd1, 1);
```

Run the sample code and the screen will display the results in the following diagram:

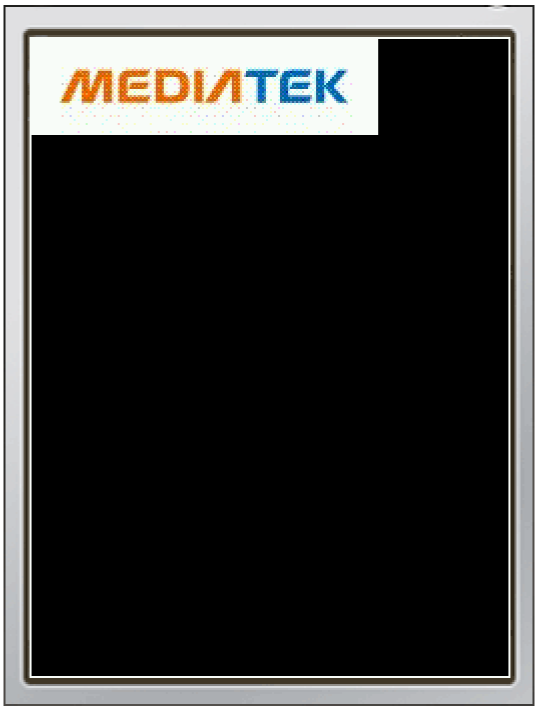


Figure 3-1. Resource demo

4 Simulator Debugging

Most of the MRE functions can be tested and debugged with the simulator. It is more convenient to perform debugging on a simulator than on a real handset.

4.1 Compiling and Building an Application

User can click on the button to start compilation of the MRE application with Visual Studio 2008.

4.2 Running the Application

User can click on the ▶ button to run the MRE application under Visual Studio 2008.

4.3 Debugging an Application

User can click on the button to debug the MRE application under Visual Studio 2008. The Debug Window and debugging tools in VC++2008 are available for use. The simulator window will start. Press the **Enter** key to turn on the simulator and start running the application as one would on a real handset.

5 Handset Debugging

5.1 Compiling and Building an Application

User can click on the !make.JPG! button to generate the final MRE application.

5.2 Running the Application

Transfer the successfully built vxp or vsm file to the handset or the \mre folder on the memory card; the MRE program can then be run on the phone.

5.3 Debugging an Application

The MRE program can be debugged on the handset using the MRE logging tool.

When necessary, the MRE program can make calls to the logging API to output log information. There are a total of 5 levels of logs available: The highest level is VM_FATAL_LEVEL, followed by VM_ERROR_LEVEL, VM_WARN_LEVEL, and VM_INFO_LEVEL, with VM_DEBUG_LEVEL being the lowest level. An application can use different API calls to output different log levels. The following is the sample code for log output:

```
VMINT val = 0;
// Update the value of val.
…
vm_log_debug(“The value = %d”, val); // Output val to the log file.
```

6 MRE Features

6.1 Basic System Characteristics

6.1.1 Multiple Applications Running Simultaneously

vm_main() is the main entry point for an MRE application. It is implemented by the MRE application, and in general, system events and callback functions for input events system are registered here by the application.
vm_exit_app() provides an exit function for the MRE application. In this function, MRE will unload the application and release all occupied resources, and it is usually the last MRE program statement, after which no other programs can be executed.

6.1.2 System Events and Input Events

After an application is started, i.e. after ***vm_main()*** has been called, MRE will immediately send the system messages **VM_MSG_CREATE** and **VM_MSG_PAINT**.

Suppose an application supports running in the background. When its window is obscured by another window or if the application is forced to exit (e.g. user pressing the END or HOME key to return to the phone's main screen), then MRE will trigger the system event **VM_MSG_HIDE**. When the obscured window disappears or when the application returns to the foreground, MRE will trigger the system event **VM_MSG_PAINT**. .

Suppose an application does not support running in the background. When its window is obscured by another window, MRE will trigger the system event **VM_MSG_INACTIVE**. When the obscured window receives a message, MRE will trigger the system event **VM_MSG_ACTIVE**.

If an application is killed, MRE will trigger the system event **VM_MSG_QUIT**.

For a list of input events supported by MRE, refer to 3.3.2.2 Key event messages and 3.3.2.3 Pen event messages. Input events must be supported by the handset.

6.1.3 Memory

MRE provides a separate heap for each application for dynamically allocated memory purposes. The memory required by the application is specified at the time the application is built and its size includes application code segment + data segment + application heap size. The application should be able to handle all situations where allocation fails.

6.1.4 File

MRE provides the following file operations, including:

Open file
Close file

Read file
Write file
Rename file
Delete file
Create directory
Delete directory
Access file attributes
Set file attributes
Get file size
Find file
Access handset or memory card drive letter
Access free disk space

6.1.5 Resources

An MRE application can utilize the external resources specified at the time the application is built. For sample usage, refer to 3.2.3 Using Resources

6.1.6 System Time

MRE can obtain system date and time from the handset.

6.1.7 Timer

MRE provides applications with two types of timer. One is used in drawing UI, and the timer stops when the LCD backlight is turned off; The other type is used for logic processing; it continues to operate even when the LCD backlight is turned off.

6.1.8 String Handling

MRE provides basic string handling functions, including UCS2 encoding format.

6.1.9 Character Set Conversion

MRE supports character string conversion between UCS2, ASCII and GB2312.

6.2 UI and Graphics

6.2.1 Graphics

MRE graphics operations are output to layers and at the same time provide a canvas for the application to maintain a buffer to improve the performance of graphics operations.

MRE graphics provide the following functions:
Performing pixel operations and drawing points, lines, rectangles and ellipses

Image decoding and painting
Layer operations
LCD operations
Text output

6.2.2 Input Methods

An MRE application enables input method functions by calling the handset's input screen. When the input screen is displayed, the MRE application window will be obscured by the input method screen. If the application supports running in the background, it will enter the background mode and receive the VM_MSG_HIDE message. If the application does not support running in the background, it will receive the VM_MSG_INACTIVE message.

The application can specify the default string and the default input method.

6.3 Multimedia

All MRE multimedia functions depend on the hardware capabilities of handsets that are eventually used to run the applications.

6.3.1 Audio

An MRE application is capable of playing audio data and audio files; it can also record audio. The audio file formats supported and the playback performance depend on the hardware capabilities of the handset running the MRE application.

6.3.2 Camera

The camera capabilities of an MRE application include previewing, taking and storing photos. The features and performance of camera depend on the hardware capabilities of the handset running the MRE application.

6.3.3 Video

An MRE application is capable of playing video data and video files, and it can also get or set certain video playback attributes. The video file formats supported and the playback performance depend on the hardware capabilities of the handset running the MRE application.

6.3.4 GPS

The GPS interface of MRE is compliant with NMEA specifications. An MRE application will be able to fix the position of the handset through access to all NMEA information. Whether GPS functions are available depends on the hardware capabilities of the handset running the MRE application.

6.3.5 Motion Sensor

An MRE application can receive notifications from motion sensor hardware. Whether motion sensor functions are available depends on the hardware capabilities of the handset running the MRE application.

6.4 Communications

6.4.1 SIM Card

MRE provides an application with a number of SIM card-related API calls to obtain relevant SIM card status and information.

6.4.2 Telephony

An MRE application can call the handset's local telephony application to make phone calls.

6.4.3 SMS

MRE provides a number of management functions for SMS, including:

- Read SMS
- Create SMS
- Delete SMS
- Send SMS
- Cancel Sending SMS
- Obtain SMS Center Number
- Register or unregister SMS Blocking

6.4.4 MMS

MRE supports sending and cancelling the sending of SMS; MMS content is sent via an XML file.

6.4.5 Phonebook

MRE provides phonebook functionalities for managing the handset's local phonebook:

- Add Contact
- Get Contact
- Update Contact
- Delete Contact
- Find Contact
- Find Contact Group
- Access Contact Memory Status

6.4.6 Cell Information

An MRE application can query cell information in order to implement cell positioning.

6.5 Network

6.5.1 Sockets

MRE provides the asynchronous BSD Sockets interface. An application obtains communications events via callback functions.

6.5.2 HTTP

MRE implements the HTTP protocol stack and complies with RFC2616 (HTTP1.1)

6.5.3 XML

MRE provides basic XML parsing capabilities. .

6.6 Logging

MRE is capable of outputting log information. For an example of using log information, refer to 5.3 Debugging an Application