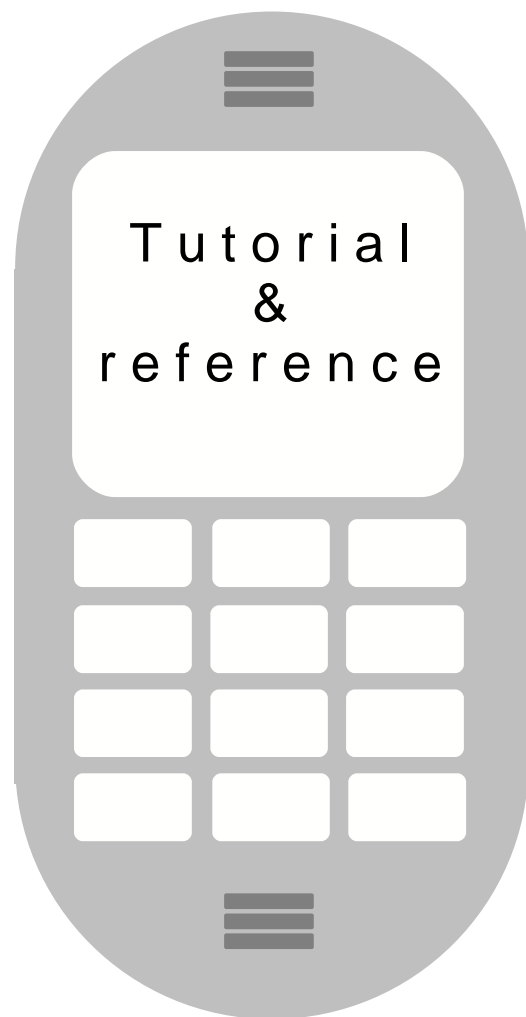


# MIDletPascal manual





# Table of Contents

Foreword	0
<b>Part I Introduction</b>	<b>4</b>
1 About MIDletPascal .....	4
2 License .....	5
3 Registration .....	8
4 Reporting bugs .....	9
<b>Part II Tutorial</b>	<b>12</b>
1 Working with integrated development environment .....	13
2 First MIDletPascal program .....	19
3 User interfaces .....	25
4 Record store .....	30
5 Connectivity .....	32
6 Advanced features .....	34
7 Performance notes .....	43
<b>Part III Procedure and Function Reference</b>	<b>46</b>
1 Drawing .....	46
2 Keypad Access .....	58
3 Date and Time .....	61
4 Math .....	66
5 String .....	72
6 Forms .....	75
7 Record Store .....	92
8 Http Connectivity .....	95
9 SMS Messaging .....	98
10 Sound and Music .....	100
11 Resource Files .....	102
12 Misc .....	104
<b>Index</b>	<b>109</b>



# Part 1

---

## Introduction



# 1 Introduction

## 1.1 About MIDletPascal

MIDletPascal is Pascal-like programming language specifically suited for development of mobile applications. MIDletPascal compiler translates Pascal code into Java™ micro edition (J2ME) bytecode. Programs written in MIDletPascal can be executed on any mobile device (such as mobile phones) supporting MIDP 1.0 and CLDC 1.0 platform.

With MIDletPascal language developing mobile applications is easy and straightforward. If you are already familiar with Pascal, Delphi or Kylix you can start writing your own mobile applications within minutes.

MIDletPascal comes with user-friendly IDE (integrated development environment) for Windows. The IDE has integrated compiler, Java bytecode preverifier and JAR archive builder so compiling and building MIDlets is as easy as a push of a button.

Since MIDletPascal generates low-level Java™ bytecode, generated MIDlets have small footprint and run fast. Some similar tools that can be found on the Internet generate an intermediate code and pack it with an interpreter into a JAR file - that approach creates large JAR files for distribution and yields slow execution.

MIDletPascal directly generates Java bytecode, so you don't need Java compiler installed on your machine, and compile process is very fast.

## 1.2 License

### **MIDletPascal End-User License Copyright 2004, 2005 Mobile Experts Group All Rights Reserved.**

Usage of this software requires compliance with this license. If you do not agree to the following terms, please do not use, install, or distribute this software.

#### **1.0 Definitions.**

1.1 For the purpose of this Agreement, "Software" shall mean the software identified above, including but not limited to any updates, modifications, revisions, copies, and associated documentation provided by Mobile Experts Group and its licensors.

#### **2.0 Grant of License.**

2.1 Evaluation Terms and Conditions. You may use this Software solely for evaluation purposes without charge for an unlimited period of time. You are free to distribute exact copies of the Software to anyone. You are prohibited from selling, or requesting donations, for any copies of the Software you distribute. You must include a copy of this license with any copy of the Software. You may not remove or alter any identification, proprietary notice, label, or trademarks which appear on or in the Software.

2.2 Non-commercial License Terms and Conditions. Use of this Software for non-commercial use (including, but not limited to personal, educational and research use) does not require purchase of a full license. Development of a commercial product using this Software terminates the Non-commercial License and you are required to obtain a Full Commercial License.

2.3 Full Commercial License Terms and Conditions. Use of the Software for non-personal use requires purchase of a full license. If you have not paid the registration fee, you may do so by visiting <http://www.midletpascal.com> on the Internet. After you have registered, you may use the Software on one or more computers, or allow multiple persons to non-simultaneously use the Software on a single computer, but not both. This is not a concurrent use license. If the Software is installed on a network server and accessed by multiple workstations, you must purchase a license for each computer on which the Software is used. You may not provide third parties access to the Software in connection with a service bureau, application service provider, or similar business, or use the Software in a business to provide file compression, decompression, or conversion services to third parties.

2.4 The Software includes the ability to create MIDlet applications. These applications archives may be freely distributed, without royalty, to anyone, as long as they have been created with registered version of MIDletPascal. Section 6 applies both to the Software and to generated MIDlet applications.

2.5 All rights of any kind in the Software which are not expressly granted in this license are entirely and exclusively reserved to and by Mobile Experts Group and its licensors.

### **3.0 Restrictions.**

3.1 The information contained in the Software is confidential and proprietary to Mobile Experts Group and its licensors, and Mobile Experts Group and its licensors retain title to all copyright, patent, trademark, trade secrets, and other proprietary rights embodied in or related to the Software.

3.2 THE SOFTWARE CANNOT BE MODIFIED. YOU MAY NOT DECOMPILE, DISASSEMBLE OR OTHERWISE ATTEMPT TO ACCESS THE SOURCE CODE OF THE SOFTWARE; EXCEPT AND ONLY TO THE EXTENT PERMITTED APPLICABLE LAW NOTWITHSTANDING THIS LIMITATION.

### **4.0 Term.**

4.1 This license is effective until terminated. Failure to comply with any provision provided in this Agreement or the use of the Software in a way not authorized or permitted under this Agreement will result in an automatic termination of this license. You agree upon such termination to destroy or dispose of the Software.

4.2 Paragraph 2.4 and Sections 3.0, 5.0, 6.0, and 7.0, including all paragraphs therein, shall survive termination this Agreement. All other rights and obligations of the Parties shall cease upon termination.

### **5.0 Upgrades, Maintenance and Support.**

5.1 During the term of this Agreement, Mobile Experts Group is under no obligation to provide upgrades, maintenance, or installation for the Software. Mobile Experts Group may provide support for the Software as governed by the documentation, and other materials related to and included with the Software. Mobile Experts Group reserves the right to revoke or modify support provided for the Software at any time, for any or no reason.

### **6.0 Limited Warranty and Remedies.**

6.1 THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY WARRANTY OF QUALITY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. MOBILE EXPERTS GROUP DOES NOT WARRANT THAT THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT IT WILL OPERATE UNINTERRUPTED OR ERROR-FREE.

6.2 NOTHING IN THIS AGREEMENT SHALL BE CONSTRUED AS A WARRANTY OR A REPRESENTATION BY MOBILE EXPERTS GROUP THAT THE SOFTWARE WILL BE FREE FROM INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS OF A THIRD PARTY. MOBILE EXPERTS GROUP HEREBY EXPRESSLY DISCLAIMS AND SHALL NOT BE RESPONSIBLE FOR ANY LIABILITY ARISING AS A RESULT OF OR IN CONNECTION WITH ANY CLAIM OR SUIT ALLEGING THAT THE DISTRIBUTION OR USE OF THE SOFTWARE INFRINGES THE RIGHT OF ANY THIRD PARTY.

6.3 IN NO EVENT WILL MOBILE EXPERTS GROUP, OR ITS LICENSORS BE



LIABLE FOR LOSS OF DATA, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES OR ANY SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES, UNDER ANY CAUSE OF ACTION AND REGARDLESS OF WHETHER OR NOT MOBILE EXPERTS GROUP, OR ITS LICENSORS, HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THIS LIMITATION WILL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN. IN ANY EVENT, MOBILE EXPERTS GROUP'S, OR ITS LICENSORS' LIABILITY ARISING OUT OF THIS AGREEMENT, THE TERMINATION THEREOF, AND/OR THE PROVISION OF GOODS OR SERVICES HEREUNDER WILL BE LIMITED TO THE GREATER OF U.S. \$1.00 OR THE AMOUNT PAID BY YOU FOR THE LICENSED SOFTWARE.

6.4 Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitations or exclusions may not apply to you.

6.5 This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

## **7.0 General.**

7.1 You agree to obtain any necessary export license or other documentation prior to exportation of the Software and you shall not export the Software or the direct product thereof in violation of the U.S. or other countries' export control laws.

7.2 The prevailing party in any legal action arising out of this Agreement shall be entitled to reimbursement for its expenses, including court costs and reasonable attorneys' fees, in addition to any other rights and remedies such party may have.

7.4 BY USING THE SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS; YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT RELATED TO THE SUBJECT MATTER HEREOF, SUPERSEDING ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN YOU AND MOBILE EXPERTS GROUP RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

## 1.3 Registration

MIDletPascal is distributed as shareware. This means that you may download and try it for free. When you are sure that it meets your needs, you should purchase a license (for more information see <http://www.midletpascal.com>).

The unregistered version does not have any time limit; however, the MIDlets created with the unregistered version display the notice that they were created using unregistered version; these MIDlets are also larger in size and run slower. Unregistered version of MIDletPascal is for personal use only.

When you purchase the license, we will send you the registration number for the MIDletPascal. The registration is valid for all version of MIDletPascal released within a year of registration.

## 1.4 Reporting bugs

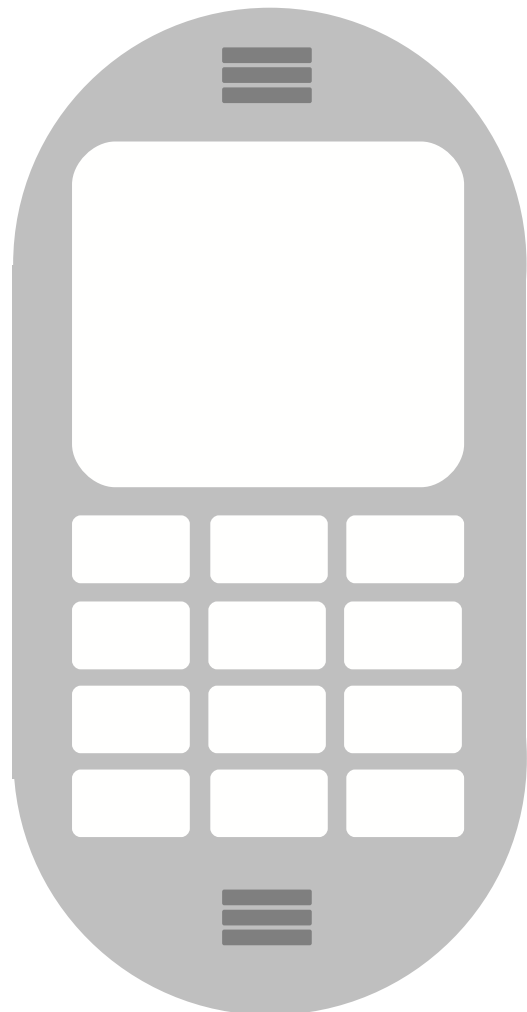
If you have discovered a bug in the program, please take your time and notify us about the problem so we can make MIDletPascal even better. Please send an email to [bugs@midletpascal.com](mailto:bugs@midletpascal.com) that describes the problem. If the problem is connected to the project you were working on, please include a ZIP file containing all project files so we can discover what caused the problem. Don't forget to mention the version of MIDletPascal you were using.



## Part 2

---

# Tutorial



## 2 Tutorial

This tutorial will help you get started with MIDletPascal programming environment and teach you the basics of writing applications for mobile phones. It assumes that you have already written programs in Pascal and that you have basic knowledge of Pascal syntax.

The tutorial contains following chapters:

**1. Working with integrated development environment ...**

Creating new project, compiling and running application, using multiple emulators, uploading the application to the phone, adding resources into the project

**2. First MIDletPascal program ...**

Drawing text on the display, reading keyboard input, drawing images, using units to separate the source

**3. User interfaces ...**

Using forms to create user interfaces, using commands, creating menus and alerts

**4. Record store ...**

Using record store for saving data

**5. Connectivity ...**

Http connectivity, SMS messaging

**6. Advanced features ...**

Custom library units, build configurations, debugging, resources

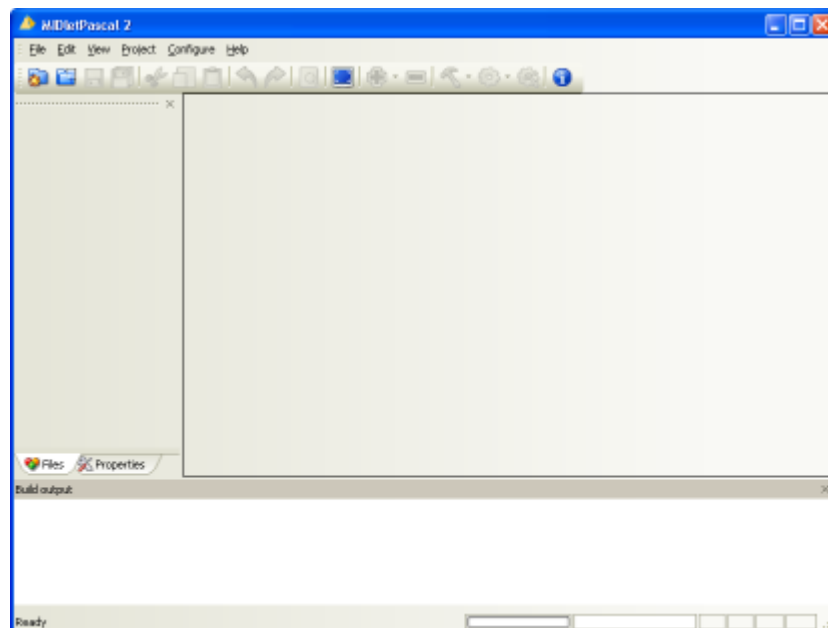
**7. Performance notes ...**

## 2.1 Working with integrated development environment

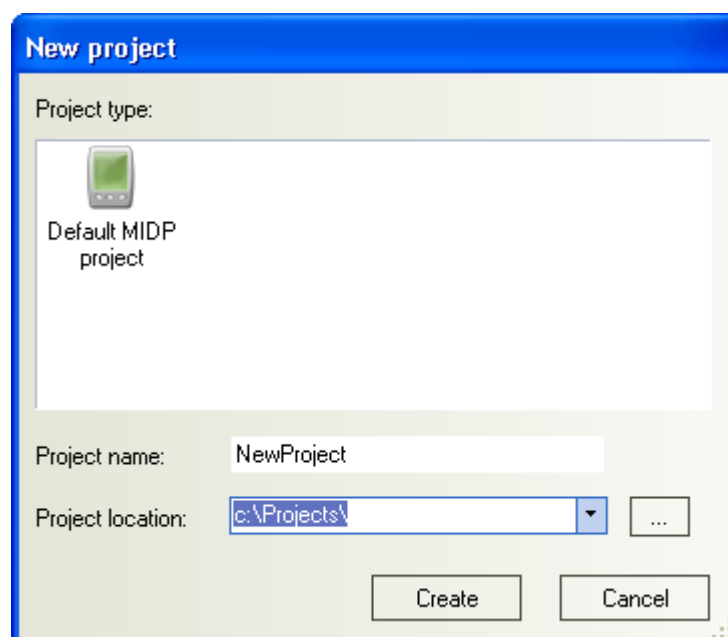
In this chapter you will learn how to create new MIDletPascal project, you will learn how to compile your project and how to run it on the emulator. You will also learn how to set up multiple emulators in MIDletPascal and how to upload your application to the mobile phone or to the WAP page. Finally, you will learn how to insert more resources into the project.

### Creating a new project

When started, MIDletPascal main application window looks like this:

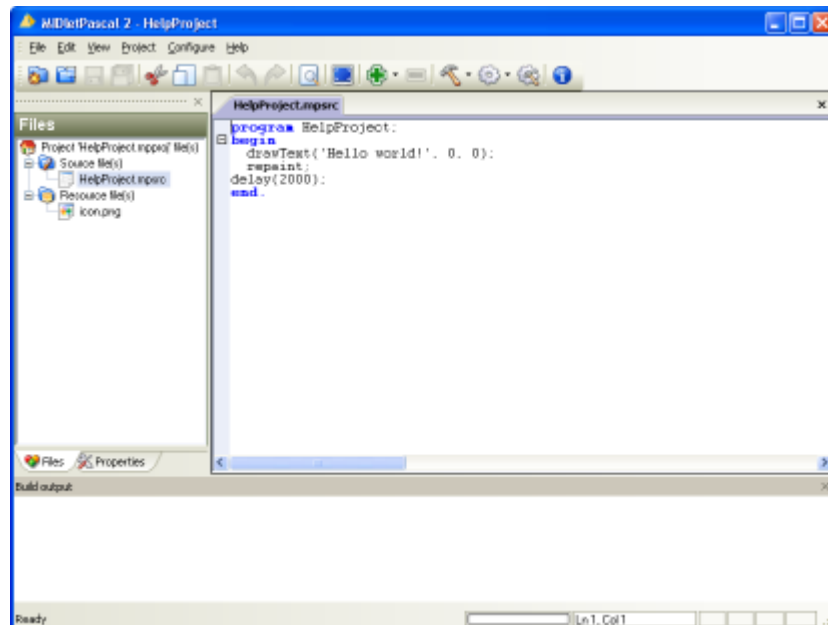


To create a new project, select the *File -> New project* menu. A dialog box appears:



You can select only one type of the project; that is, the *Default MIDP project*. Enter the project name, and select the location for the project – MIDletPascal will create a directory named <Project name> inside directory <Project location>. All project files will be stored inside the project directory.

After you click on *Create*, MIDletPascal will create the project directory and all needed project files. If all project files are created successfully, MIDletPascal window will look like this:



You have just created your first MIDletPascal project! Congratulations!

## Compiling and running the project

To compile the project, select *Project -> Build project* menu, or simply press F7 function key. MIDletPascal will start to compile your project and you will get the following output in the *Build output* bar on the bottom of the screen:

```
Build output
Build configuration: Default (MIDP 1.0, classic MIDlet)
Checking for unit dependencies...

Compiling 'helpproject.mpsrc'...
Done - 0 error(s), 0 warning(s)

Creating JAR file...
JAR file created
Creating JAD file...
JAD file created
Build successful (configuration: Default)
```

As you see, MIDletPascal tells you that there are no errors and that the executable files have been successfully created. If MIDletPascal encounters errors in the code, it will output a message explaining the error and saying in which file and on which line the error occurred. You can double-click the error message, and MIDletPascal will display the source code that caused the error so that you can correct it.



If there are no errors in the source file, MIDletPascal will create two files: the JAD file (**J**ava **A**pplication **D**escriptor) and the JAR file (**J**ava **A**rchive file with executables). Both of these files are needed by the mobile phone to run the MIDlet application. Usually, when MIDlet is downloaded on the mobile phone, a user first downloads JAD file which contains the MIDlet name and the URL of the JAR file. Then the user may select to download the JAR file with the executables.

In order to run the project, you must install a J2ME emulator on your computer. It is recommended that you install Sun Wireless Toolkit (which can be freely downloaded at the [java.sun.com](http://java.sun.com)) which contains Sun's emulator. Additionally, you can install emulators provided by the mobile phone manufacturers, such as Nokia phone emulator or a Motorola phone emulator. Setting up MIDletPascal to work with multiple emulators is explained later. By default, MIDletPascal assumes that you have installed Wireless Toolkit.

When you want to run the application, select *Project -> Run MIDlet* menu, or press the F5 function key. If you have Wireless Toolkit installed, it will bring up its J2ME emulator window:



Use the emulator to test your application.

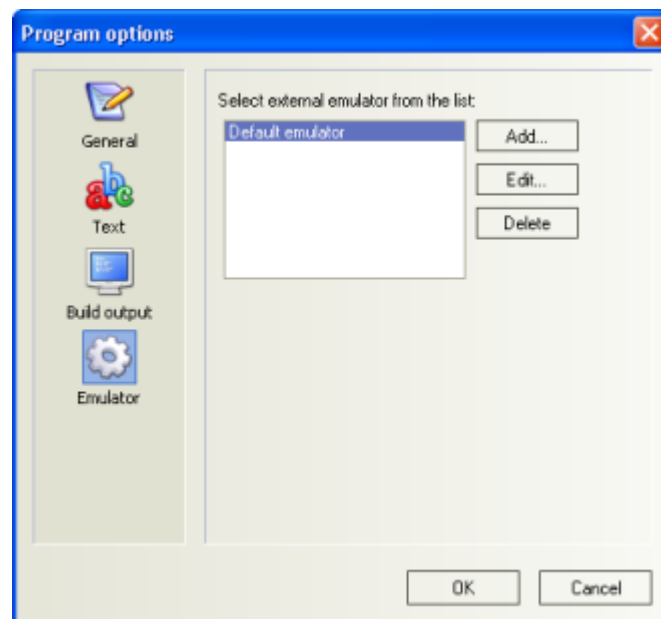
### Multiple emulators

As said before, you may want to use more than one mobile phone emulator. Although any Java executable (including those generated by MIDletPascal) should run on any Java-enabled mobile device, there are some compatibility issues. First of all, older phones support MIDP1.0 while newer (and more enhanced) devices support MIDP2.0 – so you can't run a MIDP2.0 application on a MIDP1.0 phone. Another problem is screen size: if you make your application look great on a

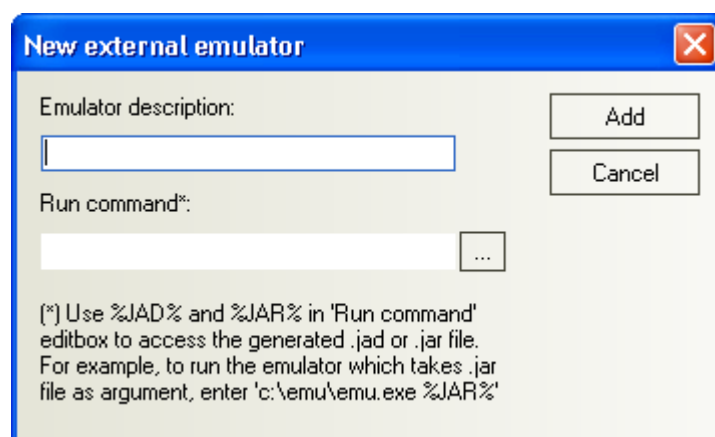
128x160 display, it may be unusable on a 128x128 display.

When developing professional MIDlet applications, you must make sure that they will run correctly on all target devices. Let's say that you want to test that your application works well both on Nokia 3510 and on Motorola V300, and you have installed both emulators on your computer. Now you have to set up MIDletPascal to work with both emulators.

Select *Configure -> Program options* menu. A dialog box will appear on the screen. Select the *Emulator* icon on the left, and you will see a following property dialog page:



Now you want to add the Nokia emulator. Select *Add...* button, and the following window will appear on the screen:



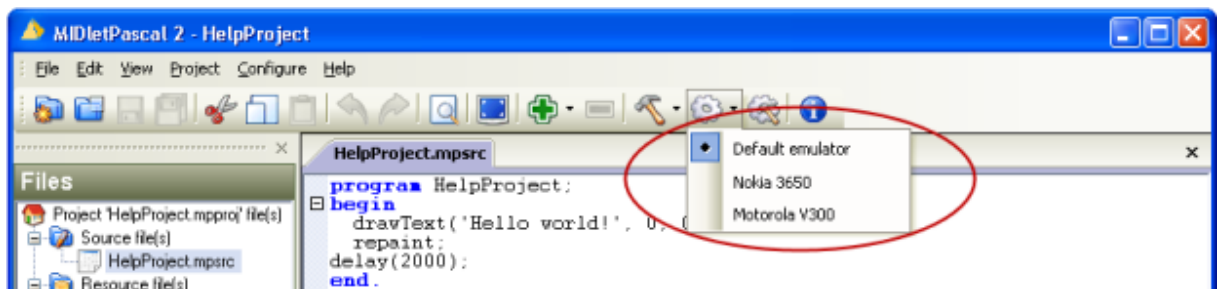
Enter the name of the emulator in the *Emulator description* field, such as *Nokia 3650*. Enter the path to the emulator executable file in the *Run command* field. The emulator executable usually takes a command-line argument that tells it which MIDlet to run. The command line argument may be the path to the JAD file or to the

JAR file. If it is the path to the JAD file, enter %JAD%, or, similarly, if it is the path to the JAR file, enter %JAR%. In case of the Nokia 3650 emulator, the *Run command* is:

```
C:\Nokia\Devices\Nokia_3510i_MIDP_SDK_v1_0\bin\3510i.exe %JAD%
```

Click the *Add* button and you will see that *Nokia 3650* appears in the list below the *Default* emulator. Then add the Motorola emulator just like you added the Nokia emulator. MIDletPascal supports up to 128 different emulators.

When you want to change the currently used emulator, click on the down arrow next to the *Run MIDlet* icon on the toolbar, and select the emulator that you want to use.



## Installing application on the mobile phone

There are two ways of installing your MIDlet application on the mobile phone: you can connect your phone to the computer, or you can connect your phone to the Internet and download the MIDlet.

The mobile phone can connect to the computer using IrDA, Bluetooth or data cable. Phone manufacturers provide software for transferring data to the phone. Refer to your phone manufacturer's manuals for detailed instructions on transferring data onto your mobile phone. You should copy two application files on the phone, the JAD and JAR file. These files can be found in the \bin subdirectory of your project's directory.

When distributing your application, you will most probably use the Internet. You will need to upload both the JAD and JAR file to a web or wap server. Anyone who wants to download the MIDlet application should download the JAD file (so all links to your application should point to the JAD file), and the mobile phone will automatically download the corresponding JAR file. However, before uploading, you should change the JAD file so that it contains the proper URL of the JAR file.

For example, let's say that you will upload your application's JAD and JAR files to <http://www.myserver.com/midlets/> location. If the application name is *Test*, you will have *Test.jar* and *Test.jad* files inside \Test\bin directory. Open the *Test.jad* file with Wordpad text editor (do not use Notepad for this!), and locate the line:

```
MIDlet-Jar-URL: Test.jar
```

Change it to contain the full URL:

```
MIDlet-Jar-URL: http://www.myserver.com/midlets/Test.jar
```

Save the changed JAD file and upload it to the server.

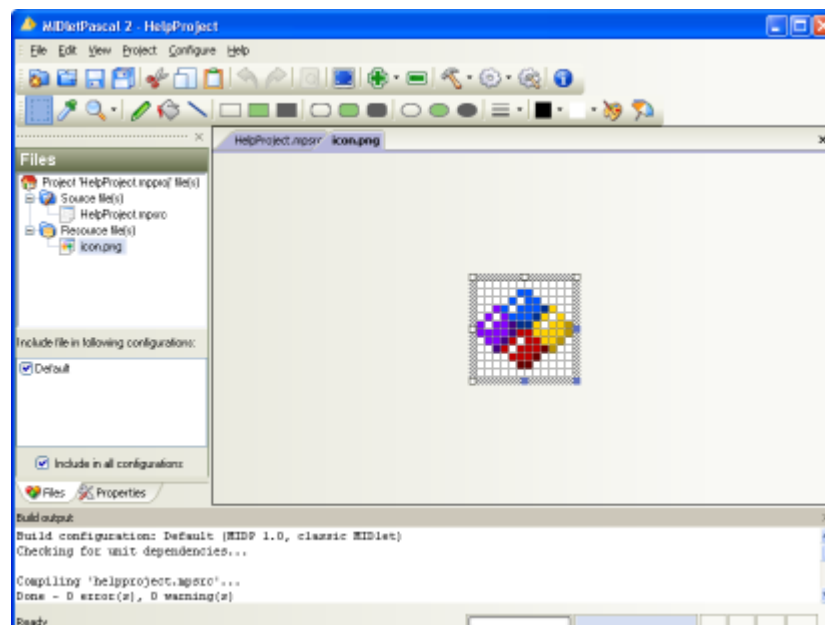
## Adding resources to the project

When you create the new MIDletPascal project, it will contain a single resource file that is used as an application icon. You may want to add more resource files such as images (in PNG file format) or sound files.

To import an existing file from your local disk into the MIDletPascal project, select *Project -> Import resource file...* menu. You will be asked to locate the file, which will be copied into the `\res` subdirectory in your project's directory. When the file is imported into the project, you should see it in the *Files* list on the left side of the MIDletPascal window.

You can also insert a new image file into the project. New image file can be created by selecting *Project -> New image resource...* menu. You will be asked for the image name and the image dimensions. Mobile Java only supports PNG images.

MIDletPascal contains a simple image editor that can be used for editing image resources. When you want to edit the image resource, double-click the resource in the *Files* list, and MIDletPascal will open the image editor:



## 2.2 First MIDletPascal program

After reading the first chapter you should know how to work with MIDletPascal environment. This chapter will show you how to write simple applications that draw on the device display, read keyboard input and use units to separate the program source code into multiple files.

### Hello world

If you have already programmed in Pascal, you would write Hello world program like this:

```
program HelloWorld;

begin
    Writeln('Hello world'); // this will not work in MIDletPascal
    !!!
end.
```

However, the procedure *Writeln* writes the text to standard output – that is the MS-DOS console window. Mobile phones don't have MS-DOS nor they have console window. Writing text on the mobile device display is more like 'drawing' that 'outputting'. To draw the text on device display, use procedure *DrawText* which takes 3 arguments: the string to write, and screen coordinates of the upper-left corner of the text bounding rectangle. Since all drawing operations are done on the memory buffer, you must call *Repaint* when you want to update the device display:

```
program HelloWorld;

begin
    DrawText('Hello world', 0, 0);
    Repaint;
end.
```

If you compile this program and try to run it, you will get the impression that the program does not get executed. In fact, the program starts, draws the text and terminates almost instantly. To see the result of the drawing operation, call procedure *Delay* which suspends the execution of the application for the given number of milliseconds:

```
program HelloWorld;

begin
    DrawText('Hello world', 0, 0);
    Repaint;
    Delay(2000);
end.
```

Procedure *Repaint* slow compared to drawing procedures such as *DrawText*. That is why *Repaint* should be called only when needed. The following example illustrates bad use of *Repaint*:

```
begin
  DrawText('Hello world', 0, 0);
  Repaint;
  DrawText('Other text', 0, 20);
  Repaint;
end.
```

The same program can be rewritten to improve performance:

```
begin
  DrawText('Hello world', 0, 0);
  DrawText('Other text', 0, 20);
  Repaint;
end.
```

## Reading from the keypad

MIDletPascal provides two functions for reading from the device keypad:

*GetKeyPressed* and *GetKeyClicked*. *GetKeyPressed* returns the key code of the key that is currently pressed. *GetKeyClicked* returns the key code of the last clicked key.

MIDletPascal has following key code constants predefined: KE\_NONE, KE\_KEY0, KE\_KEY1, KE\_KEY2, KE\_KEY3, KE\_KEY4, KE\_KEY5, KE\_KEY6, KE\_KEY7, KE\_KEY8, KE\_KEY9, KE\_STAR and KE\_POUND. These keys have the same standard key codes on all mobile phones. Unfortunately, arrow keys and other special keys have key codes that differ from one mobile phone model to the other. To get around this problem, MIDletPascal offers a function which translates the key code into an *action code*. The function is *KeyToAction*; it takes a key code as the only parameter and returns one of the following predefined constant values: GA\_NONE, GA\_UP, GA\_DOWN, GA\_LEFT, GA\_RIGHT, GA\_FIRE, GA\_GAMEA, GA\_GAMEB, GA\_GAMEC, GA\_GAMED.

The following example allows the user to move the text on the screen using the arrow keys. The program terminates when the user presses the fire or zero key.

```
program MoveText;

var x, y: integer;
    keyCode: integer;

begin
  repeat
    keyCode := GetKeyPressed; // retrieve the currently pressed
    key
    if KeyToAction(keyCode) = GA_UP then y := y - 1;
    if KeyToAction(keyCode) = GA_DOWN then y := y + 1;
    if KeyToAction(keyCode) = GA_LEFT then x := x - 1;
    if KeyToAction(keyCode) = GA_RIGHT then x := x + 1;

    SetColor(255, 255, 255); // set the drawing color to white
```

```
    FillRect(0, 0, GetWidth, GetHeight); // erase the screen
    by drawing a                          // rectangle as big
    as display

    SetColor(0, 0, 0); // set the drawing color to black
    DrawText('Hello world', x, y); // draw the text to the
    given position

    Repaint;
    Delay(100);

    until (keyCode = KE_KEY0) or (KeyToAction(keyCode) =
    GA_FIRE);

end.
```

Sometimes you might want the user to input some text. Using *GetKeyPressed* and *GetKeyClicked* for this purpose is almost impossible. User input is handled by **forms**, which are described in the next chapter.

## Drawing images

In the next example we will implement a simple *screen-saver*. The screen saver will bounce the MIDlet icon:

```
program ScreenSaver;

const up      = 1; // these constants define the direction
      down    = 0; // in which the icon moves
      left    = 1;
      right   = 0;

var x, y      : integer; // the current icon's left upper
corner position
    img       : image;   // handle of the image object
    direction_x,
    direction_y : integer; // the movement direction

begin
    img := loadImage('/icon.png');

    repeat
        // move the icon according to the current direction
        if direction_y = up      then y := y - 1;
        if direction_y = down    then y := y + 1;
        if direction_x = left    then x := x - 1;
        if direction_x = right   then x := x + 1;

        // check if the icon hit the display border
        if x = 0 then direction_x := right;
        if x = (GetWidth - GetImageWidth(img)) then direction_x :=
left;
```

```
    if y = 0 then direction_y := down;
    if y = (GetHeight - GetImageHeight(img)) then direction_y
:= up;

    SetColor(255, 255, 255);
    FillRect(0, 0, GetWidth, GetHeight);
    DrawImage(img, x, y);
    Repaint;
    Delay(50);
until GetKeyPressed <> KE_NONE;
end.
```

## Using units

When developing large programs, having only one source file is a major drawback. You cannot get around in hundreds of functions and procedures, and all the functions, procedures and global variables that you use are not grouped by their functionality. To solve this problem, you can use units just like in any standard Pascal.

There are few minor differences between MIDletPascal units and regular Pascal units. In regular Pascal, you can have as many units in a source file as you want. In MIDletPascal, you can have only one unit per file, and the file name must correspond to the unit name. MIDletPascal does not support *finalization* block that regular Pascal units can have. And finally, in MIDletPascal, units are tied to the project – if you want to use a unit from one project in another, you must create the unit in the other project, and copy the source code.

To illustrate how the units can be used in MIDletPascal, we will write a simple application that uses units.

First, create a new MIDletPascal project; let's name it *TestUnits*. Then select *Project -> New source file...* menu, and enter *Complex* as unit's name.

Enter the following code in the file *Complex.mpsrc*:

```
unit Complex;

interface

    type complex = record
        real_part, imag_part: real;
    end;

    function Add(a, b: complex): complex;
    function Sub(a, b: complex): complex;
    function MakeString(a: complex): string;

implementation

    function Add(a, b: complex): complex;
```



```
begin
  add.real_part := a.real_part + b.real_part;
  add.imag_part := a.imag_part + b.imag_part;
end;

function Sub(a, b: complex): complex;
begin
  sub.real_part := a.real_part - b.real_part;
  sub.imag_part := a.imag_part - b.imag_part;
end;

function MakeString(a: complex): string;
begin
  MakeString := ' ' + a.real_part + ' + ' + a.imag_part + 'j';
end;

end.
```

Enter the following code in file TestUnits.mpsrc:

```
program TestUnits;

uses Complex;

var x, y: Complex.complex;

begin
  x.real_part := 2; x.imag_part := 3; // x := 2 + 3j
  y.real_part := 0; y.imag_part := 1; // y := j

  x := Complex.Add(x, y);

  DrawText(Complex.MakeString(x), 0, 0);
  Repaint;
  Delay(2000);
end.
```

Units may use one another – if you want to use one unit from within the other, write

```
uses UnitName;
```

right after the `implementation` keyword. However, units may not be used circularly, that is, unit A cannot use unit B if unit B already uses unit A.

The units may contain initialization code. This is the code that is guaranteed to execute before any function, procedure or variable from the unit is accessed. The initialization code should be written after the *initialization* keyword, like in the following example:

```
unit Unit_X;

interface
  var x:integer;

implementation
```

```
// nothing to be implemented here  
  
initialization  
  x := 17;  
  
end.
```

## 2.3 User interfaces

If you want to create a simple program that asks the user to enter his/hers name, and then displays that name on the display, you need something more powerful than *GetKeyClicked* or *GetKeyPressed* functions. The more powerful mechanism is called *forms* and you can create complex user interfaces using forms. Forms are very similar to dialog boxes in windows.

### A simple form application

We'll just start by creating a simple program that uses a form to retrieve the user's name, and then writes 'Hello, <entered name>' on the screen. Here is the code for that program:

```
program HelloForm;

var itemId : integer;
    cmdNext : command;
    cmdQuit : command;
    name : string;

begin
    // switch to form mode from default canvas mode
    ShowForm;

    // add a text field to the form
    itemId := FormAddTextField('Enter your name', '', 20,
TF_ANY);

    // create a command (button)
    cmdNext := CreateCommand('Go!', CM_OK, 1);
    AddCommand(cmdNext);

    // wait until the user clickes on Go! command
    repeat until GetClickedCommand = cmdNext;

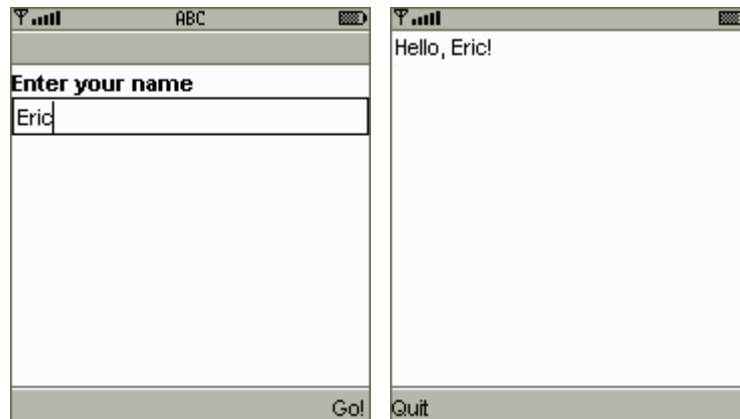
    // retrieve the entered name
    name := FormGetText(itemId);

    // switch back to canvas mode and draw the text
    ShowCanvas;

    DrawText('Hello, ' + name + '!', 0, 0);
    Repaint;

    // create a quit command
    cmdQuit := CreateCommand('Quit', CM_EXIT, 1);
    AddCommand(cmdQuit);
    repeat until GetClickedCommand = cmdQuit;
end.
```

The program produces the following form:



There are few things to be noticed. First is, you must call *ShowForm* to switch to form mode before inserting elements to form. *FormAddTextField* creates a text field and returns its identifier. The same identifier is used later to retrieve the text from the text field.

Commands are interesting because they can be used both in form mode and in canvas mode. Exception to this are canvases while using full screen mode (more on fullscreen mode in chapter 6).

### More form elements

In previous example we saw how to create a simple form. Now we will illustrate how to create a more complex form:

```
program HelloForm;

var
  stringID, textID, passID,
  imgID, gaugeID, choiceID,
  maleID, femaleID: integer;

  cmdQuit, cmdResetName, clicked : command;

begin
  ShowForm;

  SetTicker('This is a ticker control');

  // add some elements
  stringID := FormAddString('Just some title');
  textID := FormAddTextField('Enter name', 'Mr. Smith', 20,
TF_ANY);
  passID := FormAddTextField('Enter password', '', 20,
TF_PASSWORD);
  imgID := FormAddImage(LoadImage('/icon.png'));
  gaugeID := FormAddGauge('Choose your age', true, 100, 18);
  choiceID := FormAddChoice('Select your gender',
CH_EXCLUSIVE );
```

```

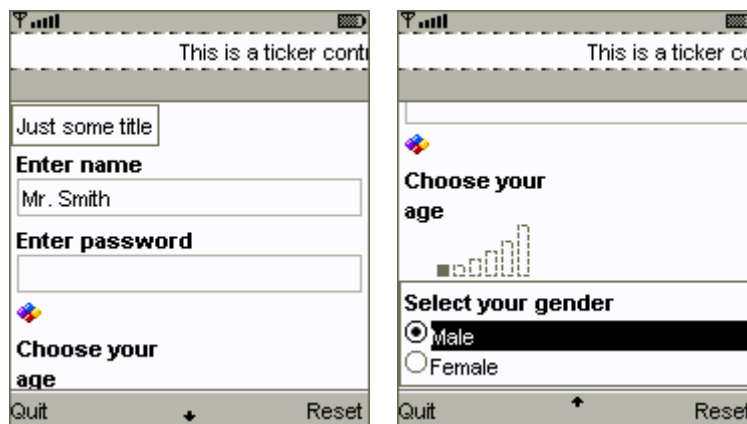
// set up the choice control
maleID := ChoiceAppendString(choiceId, 'Male');
femaleID := ChoiceAppendString(choiceId, 'Female');

// create commands
cmdQuit := CreateCommand('Quit', CM_EXIT, 1);
AddCommand(cmdQuit);
cmdResetName := CreateCommand('Reset', CM_SCREEN, 1);
AddCommand(cmdResetName);

repeat
    clicked := GetClickedCommand;
    if clicked = cmdResetName then FormSetText(textID, '');
until clicked = cmdQuit;

end.

```



To retrieve the data from the text field, use *FormGetText* function; to retrieve the value from the gauge control, use *FormGetValue* function. *ChoiceIsSelected* and *ChoiceGetSelectedIndex* functions are useful to find selected elements inside a choice list.

For more detailed information about forms, see Forms section in Procedure and function reference part of this manual.

## Menus

Beside canvas and form mode, MIDletPascal supports few more modes. One of these modes is menu mode. The following code illustrates use of menu mode:

```

program Menu;

var tetris, minesweeper, snake : integer;
    play, clicked : command;

begin
    ShowMenu('Select a game', CH_IMPLICIT);

    tetris := MenuAppendString('Tetris');

```

```
    minesweeper := MenuAppendString('Minesweeper');
    snake := MenuAppendStringImage('Snake',
LoadImage('/icon.png'));

    play := CreateCommand('Play', CM_SCREEN, 1);
    AddCommand(play);

    repeat Delay(100); until GetClickedCommand = play;

    {
        do something like this:
        if menuGetSelectedIndex = tetris then playTetris;
        if menuGetSelectedIndex = minesweeper then playMinesweeper;
        if menuGetSelectedIndex = snake then playSnake;
        ...
    }
end.
```



## Alerts

Another mode is alert mode. Alerts are simple message boxes, as illustrated in the following example program:

```
program Alert;

begin
    ShowAlert('New message',
              'You have just received a message from MrSmith',
              LoadImage('/icon.png'),
              ALERT_INFO
            );
    PlayAlertSound;

    Delay(2500);
end.
```



### Fullscreen text box

The final display mode is full screen text box. Unlike text field that is single-line, text box can contain multiple lines of text. Text box is useful when the user needs to enter large quantities of text, such as message text. The following example shows how to use full screen text box from within your application:

```
var cont : command;
    quote : string;

begin
  ShowTextBox('Enter message', '', 200, TF_ANY);
  cont := CreateCommand('Send', CM_SCREEN, 1);
  AddCommand(cont);

  repeat
    delay(100);
  until GetClickedCommand <> EmptyCommand;

  quote := GetTextBoxString;

end.
```



In this chapter full of examples you learned how to use form, alert, menu and textbox display modes to create rich user interfaces. For more detailed information on specific controls, refer to Procedure and function reference part of this manual. The next chapter will teach you how to store persistent data on the mobile device.

## 2.4 Record store

Very often an application must save some data into a persistent memory. On a computer, the data is saved to files which exist on a disk drive. Mobile phones don't have disks, so MIDlet applications need a different method of saving data. J2ME provides access to phone's flash memory through an interface called record store.

There is one very important interface between the computer file system and record store. Computer file system is unique for all applications – one application can create a file, and later, a different application can access that file. On mobile phones, each MIDlet application has its own record store space, and the spaces are isolated. In other words, there is NO way for MIDlet application to access a record store created by some other MIDlet application.

A record store is identified by its name. To open a record store, call *OpenRecordStore* function. This function takes one argument, the name of the record store, and returns an object of type *RecordStore*. If the record store with the specified name does not exist in application's record store space, a new record store is created. Record store must be closed using *CloseRecordStore* procedure. You can delete a record store (and all data inside it) by calling *DeleteRecordStore*.

Each record store contains string entries identified by their indices. This is very different from regular files that only contain raw data. You can retrieve the number of entries in a record store by calling *GetRecordStoreSize* function. Function *ReadRecordStoreEntry* takes two arguments: a record store object, and the index of the entry. It returns a string that is saved inside the record store at specific index. A new entry can be inserted into the record store by calling *AddRecordStoreEntry* function which returns the index of the newly inserted entry. *GetRecordStoreNextId* returns the index that will be given to the first element you insert using *AddRecordStoreEntry*. An entry data can be modified using *ModifyRecordStoreEntry*. Finally, an entry can be deleted from the record store by calling *DeleteRecordStoreEntry*.

The index of the first entry stored in the record store is 1. Next call to *AddRecordStoreEntry* will add an entry to index 2. The following call will add an entry to index 3, etc. If you delete an entry at index 1, and after that you call *AddRecordStoreEntry*, the new entry will be added to index 4, while the entry at index 1 will be empty. If you call *GetRecordStoreNextId*, it will return 5, and *GetRecordStoreSize* will return 3, because after deleting the first entry, there are 3 entries left in the record store.

[u reference dodati dvije nove funkcije: L(recordstore, string, int)V I Lja(recordstore)I]

The following program uses record store to display the number of times it has been run:

```
var rs      : recordStore;  
    countStr : string;  
    countInt : integer;  
    index    : integer;  
    nextId   : integer;
```



```
begin
    rs := OpenRecordStore('Count');

    nextId := GetRecordStoreNextId(rs);

    // if this is the first program run, add an entry into the
    // record store
    if nextId = 1 then
        index := AddRecordStoreEntry(rs, '0'); // the initial count
is 0

    // read the count
    countStr := ReadRecordStoreEntry(rs, 1);

    // increment count by one
    countInt := StringToInteger(countStr) + 1;
    countStr := IntegerToString(countInt);

    // save the increased count
    ModifyRecordStoreEntry(rs, countStr, 1);

    // close the record store
    CloseRecordStore(rs);

    // display the count
    ShowForm;
    index := FormAddString('Number of runs: ' + countStr);

    AddCommand(CreateCommand('Exit', CM_EXIT, 1));
    repeat Delay(100) until GetClickedCommand <> EmptyCommand;

end.
```

Note that, due to slow flash memories, accessing record store is slow operation on mobile phones. In order to improve performance, the MIDlet application should minimize reading and writing to record store.

## 2.5 Connectivity

### Http connectivity

A mobile phone can be connected to the Internet using a WAP connection. MIDlet application can initiate the connection and then send or retrieve some data on the Internet. When connecting to the Internet from your MIDlet application, be aware that mobile network operators charge Internet access and the fees are high in some countries.

All mobile phones that can connect to the Internet must support the HTTP protocol, so MIDletPascal defines functions for using HTTP protocol. Some (high-end) phones also support other protocols (such as UDP, for example). If you want to use other protocols from your MIDlet application, you will have to create your own library unit (this is explained in the next chapter).

MIDletPascal defines a type *HttpConn* that is used as a HTTP connection identifier. The following example retrieves a web page:

```
var conn          : http;
    htmlBody      : string;
    contentType   : string;

begin
    // try to open a HTTP connection to www.google.com
    if not OpenHttp(conn, 'http://www.google.com') then Halt;

    // set the HTTP method (other supported methods are HEAD and
    POST)
    SetHttpMethod(conn, GET);

    // insert a User-agent header field
    AddHttpHeader(conn, 'User-agent', 'MIDletPascal browser');

    // send HTTP request and check if the reply code is 200 (200
    = OK)
    if SendHttpMessage(conn) <> 200 then Halt;

    // read the HTTP response body
    htmlBody := GetHttpResponse(conn);

    // read Content-type response header field
    contentType := GetHttpHeader(conn, 'Content-type');

    // close the open connection
    CloseHttp(conn);
end.
```

### SMS messaging

MIDlet applications can send SMS messages automatically. MIDletPascal provides

functions for sending SMS messages. The functions are: *SmsStartSend*, *SmsIsSending* and *SmsWasSuccessfull*. The following example sends an SMS to another phone:

```
begin
  // start sending SMS to phone number +5550000
  if not SmsStartSend('sms://+5550000', 'Hello!') then Halt;

  // wait until the message is sent
  while SmsIsSending do
    Delay(100);

  // check if the message was sent successfully
  if not SmsWasSuccessfull then Halt;
end.
```

## 2.6 Advanced features

### Custom library units

MIDletPascal does not provide functions that access the complete MIDP API. However, if you need to call a MIDP API function that is not supported by MIDletPascal, you don't need to wait for the next version of MIDletPascal. You can extend the MIDletPascal functionality on your own by creating a *library unit*.

A library unit is used from your MIDletPascal program just like any other unit. To use the unit, write

```
uses unit_name;
```

at the beginning of your program. To call a function or procedure that is located inside the unit, write

```
unit_name.procedure_name(parameters...)
```

The differences between library units and regular units are the following:

- library units are written in Java, not MIDletPascal
- library units reside in `{app_install_path}\Libs` directory, not your project's directory
- library units can only contain functions and procedures, not types or variables
- functions and procedures that reside in library units can only work with integers or strings

So let's write a simple library unit. You must have Sun Java Development Kit and Sun Wireless Toolkit installed on your computer to be able to compile the library units (both can be freely downloaded from [java.sun.com](http://java.sun.com)). Let JDK register environment variables during the install process. In the following code we will assume that Wireless Toolkit is installed in `c:\wtk21\` directory.

Create the directory for your library unit, `c:\MyLib`. In this directory, create three subdirectories: `src\`, `tmpclasses\` and `classes\`. In subdirectory `src\` create the file `Lib_mylib.java` with the following contents (Java is case-sensitive unlike Pascal!):

```
// class name must be prefixed with Lib_ and all letters except
// the starting L must be lowercase
public class Lib_mylib
{
    // all functions must be static, and all letters in the
    // function name must be lowercase
    public static int square(int val)
    {
        return val * val;
    }

    public static int count_numbers(String text)
    {
```

```

        int count = 0;
        int i;
        for(i = 0; i < text.length(); i++)
        {
            if ((text.charAt(i) >= '0') && (text.charAt(i) <= '9'))
                count ++;
        }
        return count;
    }
}

```

To compile the file, open command prompt window, enter the directory c:\MyLib\src, and type (in a single line):

```

C:\MyLib\src>javac -g:none -classpath
c:\MyLib\tmpclasses;c:\WTK21\lib\midpapi20.jar;c:\WTK21\lib\cldcapi10.jar
-d c:\MyLib\tmpclasses Lib_mylib.java

```

The position to c:\MyLib directory and enter the following command:

```

C:\MyLib>c:\WTK21\bin\preverify.exe -classpath
c:\MyLib\tmpclasses;c:\WTK21\lib\midpapi20.jar;c:\WTK21\lib\cldcapi11.jar
-d c:\MyLib\classes Lib_mylib

```

The file Lib\_mylib.class found in c:\MyLib\classes directory is a library unit file that will be used by MIDletPascal. Copy in to c:\Program Files\MIDletPascal2\Libs\ directory.

Now you can use the library unit in your MIDletPascal project:

```

uses MyLib;

var idx : integer;
begin
    ShowForm;

    idx := FormAddString('5 squared is: ' + MyLib.square(5));

    idx := FormAddString('There are ' +
                        MyLib.count_numbers('a0b2c') +
                        ' numbers in ' 'a0b2c' ' ');

    Delay(5000);
end.

```

Let's say that you want to create a library unit with the functions *vibrate* and *flash\_backlight*. MIDP API functions for vibrating and flashing are called on the MIDlet's *Display* object. To retrieve the *Display* object from your library unit, use a little trick: create another file in c:\MyLib\src directory, name it FW.java, and enter the following code into the new file:

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

```
public class FW extends MIDlet
{
    public static FW fw; // the active MIDlet object
    public Display display; // the current display object
    public static Displayable CD; // the current Displayable
    object

    public void startApp()
    {}

    public void pauseApp()
    {}

    public void destroyApp(boolean unconditional)
    {}
}
```

Then add the two functions into Lib\_mylib.java:

```
public static void vibrate(int duration)
{
    FW.fw.display.vibrate(duration);
}

public static void flash_backlight(int duration)
{
    FW.fw.display.flashBacklight(duration);
}
```

Compile the two files:

```
C:\MyLib\src>javac -g:none -classpath
c:\MyLib\tmpclasses;c:\WTK21\lib\midpapi20.jar;c:\WTK21\lib\cldcapi10.jar
-d c:\MyLib\tmpclasses FW.java
```

```
C:\MyLib\src>javac -g:none -classpath
c:\MyLib\tmpclasses;c:\WTK21\lib\midpapi20.jar;c:\WTK21\lib\cldcapi10.jar
-d c:\MyLib\tmpclasses Lib_mylib.java
```

And preverify the library unit file:

```
C:\MyLib>c:\WTK21\bin\preverify.exe -classpath
c:\MyLib\tmpclasses;c:\WTK21\lib\midpapi20.jar;c:\WTK21\lib\cldcapi11.jar
-d c:\MyLib\classes Lib_mylib
```

Once again, copy the Lib\_mylib.class from c:\MyLib\classes\ directory into c:\Program Files\MIDletPascal2\Libs directory.

Now you are ready to use the new functions in your MIDletPascal program:

```
uses MyLib;

var idx : integer;
begin
```

```
MyLib.vibrate(1000);
MyLib.flash_backlight(1000);
Delay(5000);
end.
```

Final note: functions *vibrate* and *flashBacklight* are MIDP2.0 compatible. Older phones that support MIDP1.0 will crash if you try to execute the sample program. To avoid the possible crash, add exception handler to your library unit functions:

```
public static void vibrate(int duration)
{
    try
    {
        FW.fw.display.vibrate(duration);
    } catch (Exception e) { }
}

public static void flash_backlight(int duration)
{
    try
    {
        FW.fw.display.flashBacklight(duration);
    } catch (Exception e) { }
}
```

## Build configurations

Imagine that you are developing a professional MIDlet game and two mobile operators buy that game from you. Each of the operators wants that his logo is displayed when the game starts. The obvious solution is to develop a game, and, when a game is finished, create a new MIDletPascal project, copy the code from the original project and only change the logo image.

However, it is possible that a mobile operator tests your game and finds a bug. Now you have two projects where you have to fix the bug (you may fix the bug in one project, and then copy-paste the changes, but this copy-paste process may generate more bugs if you don't do it carefully).

Another similar situation may occur if your MIDlet must be optimised both for devices with 128x128 display and 220x160 display. You would create two projects, one for devices with smaller, and the other for devices with larger display. Both MIDlets would only differ in image resources and a bit of drawing code.

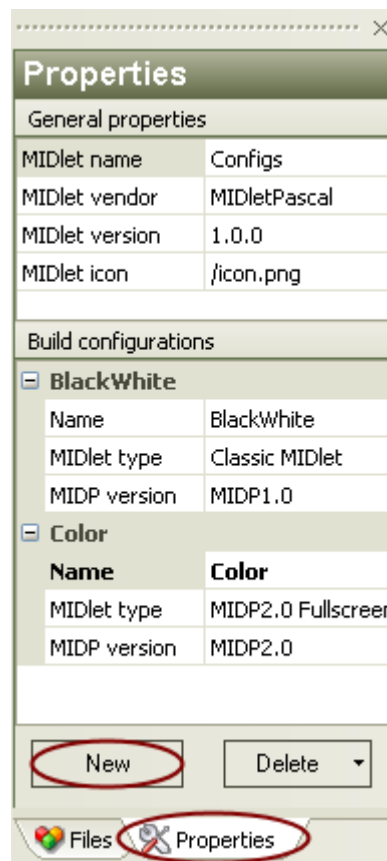
To avoid creating multiple projects, MIDletPascal offers **build configurations**. You can define multiple build configurations in your MIDletPascal project. When you build your MIDlet, you select which build configuration should be used. You can specify which resources will be included in which configurations and some parts of the source code can be compiled only for specific build configurations.

Use of build configurations will be described in a simple MIDlet application. We want

to create a MIDlet that will display an image in the center of the screen. We want to have two version of a MIDlet. The first version will display black & white image, while the other version will display a color image on a MIDP2.0 full-screen canvas.

Let's start by creating a new project, Configs. After the project has been created, select *Project -> New image resource...* and add a new image called *bw\_img.png* (make the size of the image 50x50). Draw something on the image (using black color), and save the image. Then add the new image, *color\_img.png*, and draw something on it using more colors. Save that image.

After we have created required images, we are ready to set up the build configurations. Select *Properties* tab located at the bottom of the left sidebar. One build configuration already exists, and it is called *Default*. Rename that build configuration to be called *BlackWhite*. Leave all other properties untouched. Now press the *New* button located at the bottom of the sidebar. A new build configuration will appear in the list. Change its name to *Color*, its MIDlet type to *MIDP2.0 Fullscreen* and its MIDP version to *MIDP2.0*. Save all changes to your project.



Finally you are ready to edit the source code. First write the code that will display black & white image:

```
var img: image;
begin
    img := LoadImage('/bw_img.png');
    DrawImage(img,
        (GetWidth - GetImageWidth(img))/2,
```



```
        (GetHeight - GetImageHeight(img))/2
    );
    Repaint;
    Delay(5000);
end.
```

The line with the call to `LoadImage` should be different for the second build configuration. MIDletPascal uses preprocessor statements to define which parts of code should be compiled for a given build configuration. The preprocessor statements available are:

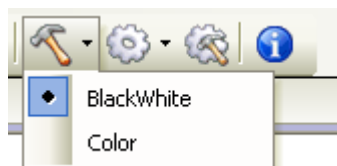
- `{ $ifdef build_configuration_name }`
- `{ $ifndef build_configuration_name }`
- `{ $else }`
- `{ $endif }`

The modified code that works with both build configurations looks like this:

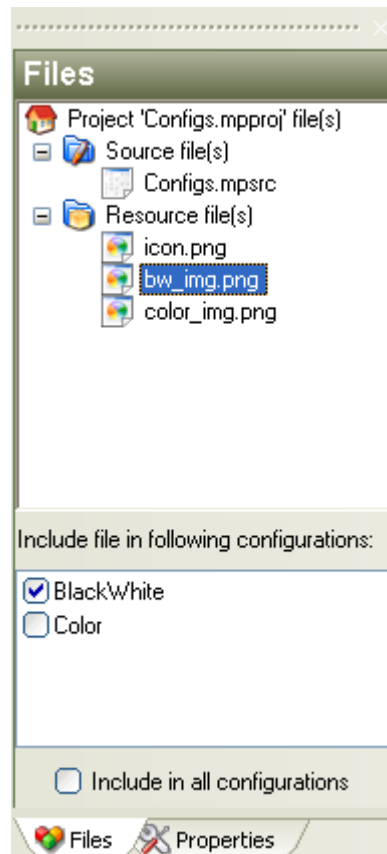
```
var img: image;
begin
    { $ifdef BlackWhite }
    img := LoadImage(' /bw_img.png' );
    { $else }
    img := LoadImage(' /color_img.png' );
    { $endif }

    DrawImage(img,
        (GetWidth - GetImageWidth(img))/2,
        (GetHeight - GetImageHeight(img))/2
    );
    Repaint;
    Delay(5000);
end.
```

To determine which configuration will be built, use the drop-down menu next to the build button:



Finally, we don't want to have `bw_img.png` in Color build configuration and vice-versa. Select back the *Files* tab, and in the files list, select the `bw_img.png`. A list with all available build configurations appears. Uncheck the Color checkbox:



Then select the `color_img.png` and uncheck the `BlackWhite` configuration.

## Debugging

MIDletPascal IDE currently does not support standard debugging. MIDletPascal, however, provides two procedures that can help in debugging your program. Both of these procedures write some text to the standard output. The standard output is available only when the application is run in the emulator; when the application is run on the device, there is no output.

You must set up your emulator to display the standard output. If you have installed Wireless Toolkit, do this as follows: open *Configure -> Program options...* menu, then select *Emulator* menu entry, click on *Add...* button and enter *Debug emulator* as emulator name and

```
C:\WTK21\bin\emulator.exe -Xdescriptor %JAD%
```

as the command string.

The first procedure is *Debug*; it takes one string parameter. This procedure simply writes the given string to the standard output.

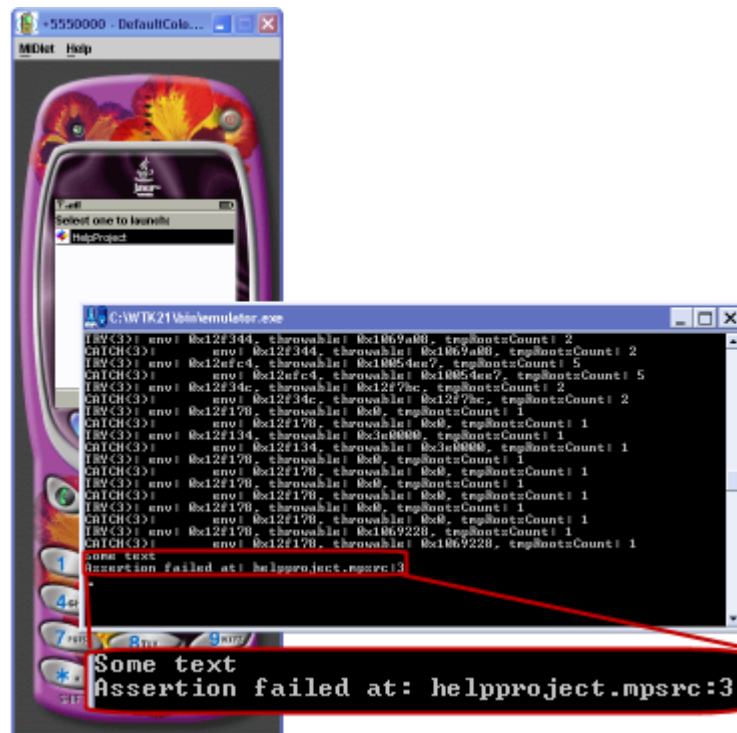
The second procedure is *Assert* and it takes a boolean value as the parameter. If that value is false, the procedure will write something like this to the standard output:

```
Assertion failed at: Tetris.mpsrc:162
```

Take a look at a simple program that uses these two procedures:

```
begin
  Debug('Some text');
  Assert(false);
end.
```

When you want to run your MIDlet application in debug mode, select Debug emulator as current emulator and run the application. A console window with the standard output will appear together with the emulator window:



## Resources

Resources are files that are included in your applications JAR file together with the executable code. Resources are usually images or music files. Sometimes it is useful to include a custom resource file into your project – for example, if you develop a dictionary application, you don't want your program to be full of *if-then* statements:

```
if englishWord = 'fish' then italianWord := 'pesce';
if englishWord = 'dog' then italianWord := 'cane';
```

This is a very bad approach for two reasons: the size of your application will be very large, and the size of your application will be very large (this is not a typo; I want to underline that the applications size will be large).

Instead, you can create a file, `dictionary.txt`, which will contain one word per line like this:

...

```
fish
pesce
dog
cane
...
```

and add this file as a resource into your project. The function that translates the word would be coded like this:

```
function EnglishToItalian(englishWord: string): string;
var italianWord : string;
    res          : resource;
    line         : string;
    pos          : integer;
begin
    res := OpenResource('/dictionary.txt'); // the name is case-
sensitive!

    if (ResourceAvailable(res)) then
    begin
        repeat
            line := ReadLine(res);

            if line = englishWord then
            begin
                italianWord := ReadLine(res);
                break;
            end;
        else
            line := ReadLine(res); // skip over italian word

        until length(line) = 0;
    end;

    englishToItalian := italianWord;
end;
```

## 2.7 Performance notes

When developing MIDlet applications be aware that processors used in mobile phones are typically 20-40 times slower than processor found in desktop computers. You should test your application on real devices (not only emulators) during development cycle to ensure that all modules execute fast enough. This short chapter lists all operations that are very slow, and gives you tips on how to make them faster.

### Large data types

Working with large data types is slow. Large data types include: arrays, records, *Image*, *Command*, *RecordStore*, *Http* and *Resource*. To improve performance, you should reduce the number of variables of these types to minimum. If possible, use global variables of these types instead of local variables. The two following programs are identical; however, the second runs faster:

```
program prog1;

procedure menu;
var cmdQuit : command;
begin
    cmdQuit := CreateCommand('Quit', CM_EXIT, 1);
    AddCommand(cmdQuit);

    repeat Delay(100); until GetClickedCommand() != EmptyCommand;
end;

begin
    menu;
end.
```

```
program prog2;
var cmdQuit : command;

procedure menu;
begin
    cmdQuit := CreateCommand('Quit', CM_EXIT, 1);
    AddCommand(cmdQuit);

    repeat Delay(100); until GetClickedCommand() != EmptyCommand;
end;

begin
    menu;
end.
```

Using two-dimensional (or higher) arrays is also slow, so use them only when they are really necessary. If you really need two-dimensional arrays, make them global (instead of local) to improve performance.

## Drawing

Drawing operations that draw into memory, such as *DrawEllipse* or *DrawText* are not slow. The operation that draws directly on to the device display, *Repaint*, is very slow operation. Be carefull to call *Repaint* only when you **really** need to update the device display.

## Real numbers

Real number operations are slow because processors used in mobile phones do not have floating-point unit. They use software emulation instead, which is very slow. Real numbers implemented in MIDletPascal have precision limited to 3 decimal places in order to make operations on real numbers faster. Try to avoid using real numbers whenever possible.

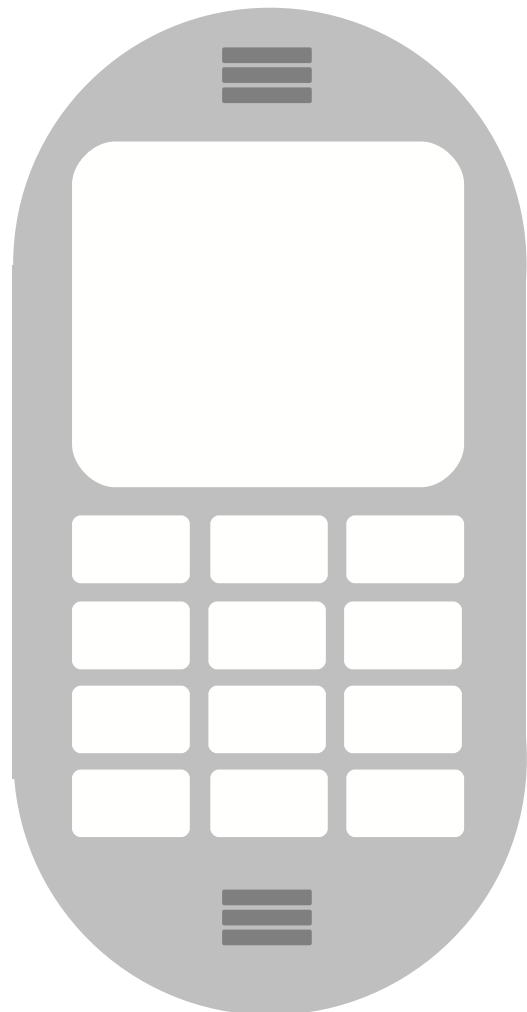
## Accessing flash file system

The flash memory used in mobile phones is slow, so all operations that read or write from/to the flash memory suffer from it. Operations that access the flash file system are the record store operations and the resource operations. It is recommended that application reads all needed data from the flash at startup, and saves its state on exit; while the application is running, it should not access the flash file system.

## Part 3

---

# Procedure and Function Reference



## 3 Procedure and Function Reference

### 3.1 Drawing

#### 3.1.1 DrawArc

Draws the arc covering the specified rectangle. The arc starts at 'startAngle' and extends for 'arcAngle' degrees. The angle of 0° degrees is at 3 o'clock position. The angle of 90° is at 12 o'clock position.

```
procedure DrawArc(x, y, width, height, startAngle, arcAngle:
integer);
```

```
begin
    DrawArc(0, 0, GetWidth, GetHeight, 0, 90);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

GetWidthGetHeightRepaint

#### 3.1.2 DrawEllipse

Draws the outline of an ellipse covering the specified rectangle.

```
procedure DrawEllipse(x, y, width, height: integer);
```

```
begin
    DrawEllipse(0, 0, GetWidth, GetHeight);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

FillEllipseGetWidthGetHeightSetColorRepaint

#### 3.1.3 DrawImage

Draws the image to the display buffer. The 'x' and 'y' coordinates are the coordinates where the upper left corner of the image will be placed.

```
procedure DrawImage(img: image; x, y: integer);
```

```
begin
    DrawImage(LoadImage('/logo.png'), 0, 0);
    Repaint;
    Delay(1000);
end.
```



MIDP1.0  
None  
LoadImageRepaint

### 3.1.4 DrawLine

Draws the line between the point ('x1', 'y1') and the point ('x2', 'y2').

```
procedure DrawLine(x1, y1, x2, y2: integer);  
  
begin  
    DrawLine(10, 15, 25, 35);  
    Repaint;  
    Delay(1000);  
end.
```

MIDP1.0  
None  
SetColorRepaint

### 3.1.5 DrawRect

Draws the outline of the specified rectangle.

```
procedure DrawRect(x, y, width, height: integer);  
  
begin  
    DrawRect(5, 5, 20, 20);  
    Repaint;  
    Delay(1000);  
end.
```

MIDP1.0  
None  
FillRectSetColorRepaint

### 3.1.6 DrawRoundRect

Draws the outline of the specified rounded corner rectangle.

```
procedure DrawRoundRect(x, y, width, height, arcWidth,  
    arcHeight: integer);  
  
begin  
    DrawRoundRect(5, 5, 20, 20, 2, 2);  
    Repaint;  
    Delay(1000);  
end.
```

MIDP1.0

None

FillRoundRectSetColorRepaint

### 3.1.7 DrawText

Draws the text 'text' to the display buffer. The 'xPos' and 'yPos' are coordinates of the upper left corner of the text-bounding box.

```
procedure DrawText(text: string, xPos, yPos: integer);
```

```
begin
    SetColor(255, 0, 0);
    DrawText('Hello world', 0, 0);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

GetStringWidthGetStringHeightSetFontSetDefaultFontSetColorRepaint

### 3.1.8 FillEllipse

Fills the ellipse covering the specified rectangle.

```
procedure FillEllipse(x, y, width, height: integer);
```

```
begin
    FillEllipse(0, 0, GetWidth, GetHeight);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

DrawEllipseGetWidthGetHeightSetColorRepaint

### 3.1.9 FillRect

Fills the specified rectangle.

```
procedure FillRect(x, y, width, height: integer);
```

```
begin
    FillRect(5, 5, 20, 20);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

DrawRectSetColorRepaint

### 3.1.10 FillRoundRect

Fills the specified rounded corner rectangle.

```
procedure FillRoundRect(x, y, width, height, arcWidth,
    arcHeight: integer);
```

```
begin
    FillRoundRect(5, 5, 20, 20, 2, 2);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

DrawRoundRectSetColorRepaint

### 3.1.11 GetColorBlue

Returns the blue component value of the current color.

```
function GetColorBlue: integer;
```

```
begin
    SetColor(0, 0, 127);
    if GetColorBlue <> 127 then
    begin
        DrawText('This should never happen', 0, 0);
        Repaint;
    end;
end.
```

```
        end;  
        Delay(1000);  
    end.
```

MIDP1.0

None

SetColorGetColorRedGetColorGreen

### 3.1.12 GetColorGreen

Returns the green component value of the current color.

```
function GetColorGreen: integer;
```

```
begin  
    SetColor(0, 127, 0);  
    if GetColorGreen <> 0 then  
    begin  
        DrawText('This should never happen', 0, 0);  
        Repaint;  
    end;  
    Delay(1000);  
end.
```

MIDP1.0

None

SetColorGetColorRedGetColorBlue

### 3.1.13 GetColorRed

Returns the red component value of the current color.

```
function GetColorRed: integer;
```

```
begin  
    SetColor(127, 0, 0);  
    if GetColorRed <> 127 then  
    begin  
        DrawText('This should never happen', 0, 0);  
        Repaint;  
    end;  
    Delay(1000);  
end.
```

MIDP1.0  
None  
SetColorGetColorGreenGetColorBlue

### 3.1.14 GetColorsNum

Returns the number of different colors that the device display can show.

```
function GetColorsNum: integer;
```

MIDP1.0  
None  
IsColorDisplay

### 3.1.15 GetHeight

Returns the height (in pixels) of the display area.

```
function GetHeight: integer;
```

```
begin
    DrawEllipse(0, 0, GetWidth, GetHeight);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0  
None  
GetWidth

### 3.1.16 GetImageHeight

Returns the height of the given image in pixels.

```
function GetImageHeight(img: image): integer;
```

MIDP1.0  
None  
LoadImageGetImageWidthDrawImage

### 3.1.17 GetImageWidth

Returns the width of the given image in pixels.

```
function getImageWidth(img: image): integer;
```

```
begin
    DrawEllipse(0, 0, GetWidth, GetHeight);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

LoadImageGetImageHeightDrawImage

### 3.1.18 GetStringHeight

Returns the height (in pixels) for showing the 'text' on display in the current font.

```
function GetStringHeight(text: string): integer;
```

```
var text: string;
    height: integer;
begin
    text := 'Text to display';
    height := GetStringHeight(text);
    DrawText(text, 0, (GetHeight - height)/2);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

GetStringWidthGetWidthGetHeightRepaint

### 3.1.19 GetStringWidth

Returns the width (in pixels) for showing the 'text' on display in the current font.

```
function GetStringWidth(text: string): integer;
```

```
var text: string;
    width: integer;
begin
    text := 'Text to display';
    width := GetStringWidth(text);
```

```
    DrawText(text, (GetWidth - width)/2, 0);  
    Repaint;  
    Delay(1000);  
end.
```

MIDP1.0  
None  
GetStringHeightGetWidthGetHeightRepaint

### 3.1.20 GetWidth

Returns the width (in pixels) of the display area.

```
function GetWidth: integer;  
  
begin  
    DrawEllipse(0, 0, GetWidth, GetHeight);  
    Repaint;  
    Delay(1000);  
end.
```

MIDP1.0  
None  
GetHeight

### 3.1.21 IsColorDisplay

Returns true if the device display can show colors; otherwise returns false.

```
function IsColorDisplay: boolean;
```

MIDP1.0  
None  
GetColorsNum

### 3.1.22 LoadImage

Loads the image from the resource and returns an 'image' object. To access the file inside the JAR archive (the file added as a resource into the MIDletPascal project), the 'resource' parameter must begin with '/' character. If the resource name is invalid, the application will fail.

```
function LoadImage(resource: string): image  
  
begin  
    DrawImage(LoadImage('/icon.png'), 0, 0);
```

```
    Repaint;  
    Delay(1000); { wait 1 second before the MIDlet terminates }  
end.
```

MIDP1.0

None

DrawImageGetImageWidthGetImageHeight

### 3.1.23 Plot

Sets the color of a single pixel to the current color.

```
procedure Plot(x, y:integer);
```

```
begin  
    SetColor(255, 0, 0);  
    Plot(5, 10);  
    Repaint;  
    Delay(1000);  
end.
```

MIDP1.0

None

SetColorRepaint

### 3.1.24 Repaint

Repaints the device display. All drawing functions (such as drawLine, drawText, fillRect etc.) do not draw directly to the device display. The drawing functions draw to the off-screen buffer, and the procedure 'repaint' copies the off-screen buffer to the device display. The repainting is a time-consuming procedure so it should be called as rarely as possible.

```
procedure Repaint;
```

```
begin  
    DrawText('Hello world', 0, 0);  
    Repaint;  
    Delay(1000);  
end.
```

MIDP1.0

None



### 3.1.25 SetClip

Sets the clipping area for the subsequent drawing operations. All drawing operations which take place outside of the clipping area rectangle will be omitted.

```
procedure SetClip(int x, int y, int width, int height);
```

```
begin
    // draw the ellipse
    SetColor(0, 0, 0);
    FillEllipse(0, 0, GetWidth, GetHeight);
    Repaint;
    Delay(2000);

    // draw ellipse again with setClip called before
    SetClip(10, 10, 15, 25);
    FillEllipse(0, 0, GetWidth, GetHeight);
    Repaint;
    Delay(2000);
end.
```

MIDP1.0

None

FillEllipseGetWidthGetHeightSetColorRepaint

### 3.1.26 SetColor

Set the color to be used for drawing all the subsequent graphical elements. The 'red', 'green' and 'blue' components can take values between 0 and 255. For example, the black color representation is (0, 0, 0), bright red is (255, 0, 0), and white is (255, 255, 255).

```
procedure SetColor(red, green, blue:integer);
```

```
begin
    SetColor(0, 0, 255); { set the color to bright blue }
    DrawText('Hello world', 0, 0);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0

None

GetColorRedGetColorGreenGetColorBlue

### 3.1.27 SetDefaultFont

Sets the current font to the default font used on the device. Note that some devices use only one font, so calling `setDefaultFont` will have no effect on those devices.

```
procedure SetDefaultFont;
```

MIDP1.0  
None  
SetFont

### 3.1.28 SetFont

Sets the font to be used for displaying text.

The 'fontFace' can be any of the following predefined constants:

- FONT\_FACE\_SYSTEM
- FONT\_FACE\_MONOSPACE
- FONT\_FACE\_PROPORTIONAL

The 'fontStyle' can be any of the following predefined constants:

- FONT\_STYLE\_PLAIN
- FONT\_STYLE\_BOLD
- FONT\_STYLE\_ITALIC
- FONT\_STYLE\_UNDERLINE

These styles may be combined together using the logical operator `or`. For exaple, to create the font that is bold and underlined, use `(FONT_STYLE_BOLD or FONT_STYLE_UNDERLINE)` as the value for the font style.

The font size can be any of the following predefined constants:

- FONT\_SIZE\_SMALL
- FONT\_SIZE\_MEDIUM
- FONT\_SIZE\_LARGE

Note that some devices use only one font, so calling `setFont` will have no effect on those devices.

```
procedure SetFont(fontFace, fontStyle, fontSize);
```

```
begin
    SetFont(FONT_FACE_SYSTEM,
            FONT_STYLE_BOLD or FONT_STYLE_UNDERLINE,
            FONT_SIZE_LARGE);
    DrawText('Hello world', 0, 0);
    Repaint;
    Delay(1000);
end.
```

MIDP1.0  
None

## SetDefaultFontSetColorRepaint

## 3.2 Keypad Access

### 3.2.1 GetKeyClicked

Returns the code of the last clicked key on the keypad, or KE\_NONE if no key is pressed. The standard key codes have predefined constant values:

- KE\_KEY0
- KE\_KEY1
- KE\_KEY2
- KE\_KEY3
- KE\_KEY4
- KE\_KEY5
- KE\_KEY6
- KE\_KEY7
- KE\_KEY8
- KE\_KEY9
- KE\_STAR
- KE\_POUND

```
function GetKeyClicked: integer;

begin
    while GetKeyClicked <> KE_STAR do
        begin
            Delay(100);
        end;
    end.

MIDP1.0
None
GetKeyPressedKeyToAction
```

### 3.2.2 GetKeyPressed

Returns the code of the key that is currently pressed, or KE\_NONE if no key is pressed. The standard key codes have predefined constant values:

- KE\_KEY0
- KE\_KEY1
- KE\_KEY2
- KE\_KEY3
- KE\_KEY4
- KE\_KEY5
- KE\_KEY6
- KE\_KEY7

- KE\_KEY8
- KE\_KEY9
- KE\_STAR
- KE\_POUND

```
function GetKeyPressed: integer;
```

```
begin
  while GetKeyPressed <> KE_STAR do
    begin
      Delay(100);
    end;
  end.
```

MIDP1.0

None

GetKeyClickedKeyToAction

### 3.2.3 KeyToAction

getKeyClicked and getKeyPressed functions return the key code of the key pressed. However, different devices have different key mappings. For example, one device may have FIRE key return key code 100, and the other device may have its FIRE key return key code 120. To avoid this problem, you can use the keyToAction function that translates the key code into mapped game actions. keyToAction can return any of the following predefined values:

- GA\_NONE
- GA\_UP
- GA\_DOWN
- GA\_LEFT
- GA\_RIGHT
- GA\_FIRE
- GA\_GAMEA
- GA\_GAMEB
- GA\_GAMEC
- GA\_GAMED

```
function KeyToAction(keyCode: integer): integer;
```

```
begin
  while KeyToAction(getKeyClicked) <> GA_FIRE do
    begin
      Delay(100);
    end;
```

end.

MIDP1.0

None

GetKeyPressedGetKeyClicked

### 3.3 Date and Time

#### 3.3.1 Delay

Suspends the execution of a program for the given time in milliseconds. To display a message on to the screen, wait 2 seconds, and then close MIDlet, use the following code:

```
procedure Delay(millis: integer);
```

```
begin
    DrawText('Hello world', 0, 0);
    Repaint;
    Delay(2000);
end.
```

MIDP1.0

None

#### 3.3.2 GetCurrentTime

Returns the current time in seconds since midnight 1.1.1970. The returned value (the number of seconds) can be used as argument to call functions such as `GetMonth` or `GetHour`.

```
function GetCurrentTime: integer;
```

```
var time: integer;
    text: string;
begin
    time := GetCurrentTime;
    text := 'Current time is ' + GetHour(time);
    text := text + ':' + GetMinute(time);
    text := text + ':' + GetSecond(time);
    DrawText(text, 0, 0);
    Repaint;
    Delay(1000); { wait 1 second before MIDlet terminates }
end.
```

MIDP1.0

None

`GetYearGetMonthGetDayGetHourGetMinuteGetSecondGetWeekDayGetYearDay`

### 3.3.3 GetDay

Returns the day in the month for the given time (time is represented in seconds since 1.1.1970. and can be retrieved by calling `GetCurrentTime` function). The returned value is between 1 and 31.

```
function GetDay(time: integer): integer;
```

MIDP1.0

None

`GetCurrentTime``GetYear``GetMonth``GetWeekDay``GetYearDay`

### 3.3.4 GetHour

Returns the hour for the given time (time is represented in seconds since midnight 1.1.1970 and can be retrieved by calling `GetCurrentTime` function). The returned value is between 0 and 23.

```
function GetHour(time: integer): integer;
```

```
var time: integer;  
    text: string;  
begin  
    time := GetCurrentTime;  
    text := 'Current time is ' + GetHour(time);  
    text := text + ':' + GetMinute(time);  
    text := text + ':' + GetSecond(time);  
    DrawText(text, 0, 0);  
    Repaint;  
    Delay(1000); { wait 1 second before MIDlet terminates }  
end.
```

MIDP1.0

None

`GetCurrentTime``GetMinute``GetSecond`

### 3.3.5 GetMinute

Returns the minute for the given time (time is represented in seconds since midnight 1.1.1970 and can be retrieved by calling `GetCurrentTime` function). The returned value is between 0 and 59.

```
function GetMinute(time: integer): integer;
```



```
var time: integer;
    text: string;
begin
    time := GetCurrentTime;
    text := 'Current time is ' + GetHour(time);
    text := text + ':' + GetMinute(time);
    text := text + ':' + GetSecond(time);
    DrawText(text, 0, 0);
    Repaint;
    Delay(1000); { wait 1 second before MIDlet terminates }
end.
```

MIDP1.0

None

GetCurrentTimeGetHourGetSecond

### 3.3.6 GetMonth

Returns the month number for the given time (time is represented in seconds since 1.1.1970. and can be retrieved by calling `GetCurrentTime` function). The returned value is between 1 and 12.

```
function GetMonth(time: integer): integer;
```

MIDP1.0

None

GetCurrentTimeGetYearGetDay

### 3.3.7 GetRelativeTimeMs

Return the current time in milliseconds. Note that the return value is 32-bit integer that can represent only  $2^{32}$  milliseconds, which is a little bit over 48 days. So every 48 days, this value is reset and starts counting from 0. Do not use this function to determine the current date, but it can be used for implementing timers in an application.

Consider a simple game (such as Tetris, for example) that needs to move the block down each second. Also, the game should move the block left or right when the user presses the left/right key on the keypad. The main program loop could be implemented as follows:

```
function GetRelativeTimeMs: integer;
```

```
...
lastSavedTime := GetRelativeTimeMs; { initialize the timer }
repeat
    { read and process the keypad input }
```

```
key := GetKeyClicked;
if KeyToAction(key) = GA_LEFT then moveLeft;
if KeyToAction(key) = GA_RIGHT then moveRight;

{ check if 1 second has passed }
if ((GetRelativeTimeMs - lastSavedTime) > 1000)
  or (GetRelativeTimeMs < lastSavedTime) { check if the
timer is reset after 48 days }
then
begin
  lastSavedTime := GetRelativeTimeMs;
  moveDown;
end;
until gameOver;
...
```

MIDP1.0

None

### 3.3.8 GetSecond

Returns the second for the given time (time is represented in seconds since midnight 1.1.1970 and can be retrieved by calling `GetCurrentTime` function). The returned value is between 0 and 59.

```
function GetSecond(time: integer): integer;
```

```
var time: integer;
    text: string;
begin
  time := GetCurrentTime;
  text := 'Current time is ' + GetHour(time);
  text := text + ':' + GetMinute(time);
  text := text + ':' + GetSecond(time);
  DrawText(text, 0, 0);
  Repaint;
  Delay(1000); { wait 1 second before MIDlet terminates }
end.
```

MIDP1.0

None

`GetCurrentTime``GetHour``GetMinute`

### 3.3.9 GetWeekDay

Returns the day in week for the given time (time is represented in seconds since 1.1.1970. and can be retrieved by calling `GetCurrentTime` function). Returned value is 1 for Sunday, 2 for Monday, etc (7 is Saturday).

```
function GetWeekDay(time: integer): integer;
```

MIDP1.0

None

`GetCurrentTime``GetYear``GetMonth``GetDay``GetYearDay`

### 3.3.10 GetYear

Returns the year for the given time (time is represented in seconds since 1.1.1970. and can be retrieved by calling `GetCurrentTime` function).

```
function GetYear(time: integer): integer;
```

MIDP1.0

None

`GetCurrentTime``GetMonth``GetDay`

### 3.3.11 GetYearDay

Returns the day in year for the given time (time is represented in seconds since midnight 1.1.1970. and can be retrieved by calling `GetCurrentTime` function). Returned value is between 1 and 366.

```
function GetYearDay(time: integer): integer;
```

MIDP1.0

None

`GetCurrentTime``GetYear``GetMonth``GetDay``GetWeekDay`

## 3.4 Math

### 3.4.1 Abs

Returns the absolute value of the given number.

```
function Abs(n: integer): integer;
```

MIDP1.0  
None

### 3.4.2 Acos

Returns the arc cosine of a number, in the range 0 to pi (in radians).

```
function Acos(num: real):real;
```

MIDP1.0  
None

### 3.4.3 Asin

Returns the arc sine of a number, in the range -pi/2 to the pi/2 (in radians).

```
function Asin(num: real):real;
```

MIDP1.0  
None

### 3.4.4 Atan

Returns the arc tangent, in the range from  $-\pi/2$  to  $\pi/2$  (represented in radians).

```
function Atan(num: real):real;
```

MIDP1.0  
None

### 3.4.5 Atan2

Converts the rectangular coordinates (x,y) into polar coordinates (r, theta). This method computes the phase theta by computing an arc tangent of y/x in the range of  $-\pi$  to  $\pi$ .

```
function Atan2(y,x: real):real;
```

MIDP1.0  
None

### 3.4.6 Cos

Returns the cosine of the given number (represented in radians).

```
function Cos(num: real):real;
```

MIDP1.0  
None

### 3.4.7 Exp

Returns the Euler's number 'e' raised to the value of 'num'.

```
function Exp(num: real):real;
```

MIDP1.0

None

### 3.4.8 Frac

Returns the fraction of a number (the fraction of 1.234 is 0.234).

```
function Frac(num: real):real;
```

MIDP1.0

None

### 3.4.9 Log

Returns the natural logarithm of the given number.

```
function Log(num: real):real;
```

MIDP1.0

None

### 3.4.10 Log10

Returns the logarithm base 10 of the given number.

```
function Log10(num: real):real;
```

MIDP1.0

None

### 3.4.11 Pow

Returns the value of 'a' raised to the power of 'b'.

```
function Pow(a, b: real):real;
```

MIDP1.0  
None

### 3.4.12 Rabs

Returns the absolute value of the real number.

```
function Rabs(num:real):real;
```

MIDP1.0  
None

### 3.4.13 Sin

Returns the sine of the given number (represented in radians).

```
function Sin(num: real):real;
```

MIDP1.0  
None

### 3.4.14 Sqr

Returns the square of the given number.

```
function Sqr(n: integer): integer;
```

MIDP1.0  
None

### 3.4.15 Sqrt

Returns the square root of the given number.

```
function Sqrt(num: real):real;
```

MIDP1.0  
None

### 3.4.16 Tan

Returns the tangent of an angle represented in radians.

```
function Tan(num: real):real;
```

MIDP1.0  
None

### 3.4.17 ToDegrees

Converts the angle from radians into degrees.

```
function ToDegrees(num: real):real;
```

MIDP1.0  
None



### 3.4.18 ToRadians

Converts the angle from degrees into radians.

```
function ToRadians(num: real):real;
```

MIDP1.0

None

### 3.4.19 Trunc

Truncates the real number and returns only the integer part.

```
function Trunc(num: real):integer;
```

MIDP1.0

None

## 3.5 String

### 3.5.1 Copy

The function returns the substring of the given string which starts at position 'begin' and ends at position 'end'-1. For example, `copy('MIDletPascal', 2, 5)` returns 'Dle'. **Note that this is different from the classic Pascal function copy.**

Strings in MIDletPascal are slightly different than strings in regular Pascal: the index of the first character in string is 0 in MIDletPascal.

```
function Copy(str1: string; begin, end: integer): string;
```

MIDP1.0  
None  
LengthPos

### 3.5.2 GetChar

Returns the character at a given index within the string. The first character in a string is at index 0. If 'pos' is larger than the length of the string, character with ASCII code '0' is returned.

```
function GetChar(str: string; pos: integer): char;
```

MIDP1.0  
None  
SetChar

### 3.5.3 IntegerToString

Converts integer into string representation of the same number.

```
function IntegerToString(val: integer): string;
```

```
var i: integer;  
    s: string;  
begin  
    i := 15;  
    s := IntegerToString(i);  
    s := ' ' + i; // this has the same effect as the previous  
statement  
end.
```

MIDP1.0  
None

### 3.5.4 Length

Returns the length of the given string. Strings in MIDletPascal are slightly different than strings in regular Pascal: the index of the first character in string is 0 in MIDletPascal.

```
function Length(str: string): integer;
```

MIDP1.0  
None  
CopyPos

### 3.5.5 Locase

Returns the copy of the 'str' string with all letters in lowercase.

```
function Locase(str: string): string;
```

MIDP1.0  
None  
UppcaseCopy

### 3.5.6 Pos

Returns the position of the first occurrence of 'str2' in 'str1', or -1 if 'str2' does not occur in 'str1'. The comparison is case-sensitive.

```
function Pos(str1, str2: string): integer;
```

MIDP1.0  
None  
CopyLength

### 3.5.7 SetChar

Returns the string that is identical to 'str' except that it has character 'c' at position 'pos'. Position indexes start with zero. If 'pos' is larger than the length of the 'str', a copy of 'str' is returned.

```
function SetChar(str: string; c: char; pos: integer): string;
```

MIDP1.0  
None  
GetChar

### 3.5.8 StringToInteger

Transforms the string into an integer (base 10 is used). If the string 's' is not a valid integer, the function will return 0. The string 's' may contain only digits, except for the first character which may be a '+' or a '-' sign.

```
function StringToInteger(s:string):integer;
```

MIDP1.0  
None

### 3.5.9 StringToReal

Transforms the string into the real number. The second parameter is the base for transformation.

```
function StringToReal(str:string; base:integer):real;
```

MIDP1.0  
None

### 3.5.10 Uppcase

Returns the copy of the 'str' string with all letters capitalized.

```
function Uppcase(str: string): string;
```

MIDP1.0  
None  
LocaseCopy

## 3.6 Forms

### 3.6.1 AddCommand

Inserts a command on the device display. Some devices (such as Motorola mobile phones) will not draw the command until `Repaint` is called. So after calling `AddCommand` be sure to call `Repaint`.

```
procedure AddCommand(cmd: command);
```

MIDP1.0

None

CreateCommandGetClickedCommandRemoveCommandEmptyCommand

### 3.6.2 ChoiceAppendString

Inserts a string into the choice group element identified by choice group ID. The function returns the index of the newly inserted element within the choice group.

```
function ChoiceAppendString(choiceID: integer;  
itemText:string):integer;
```

```
var choiceGroupID: integer;  
    NY, LA: integer;  
begin  
    ShowForm;  
    choiceGroupID := FormAddChoice('Where do you live?',  
CF_EXCLUSIVE);  
    NY := ChoiceAppendString(choiceGroupID, 'New York');  
    LA := ChoiceAppendString(choiceGroupID, 'Los Angeles');  
end.
```

MIDP1.0

None

FormAddChoiceChoiceAppendStringImageChoiceIsSelectedChoiceGetSelectedIndex

### 3.6.3 ChoiceAppendStringImage

Inserts a string and an image into the choice group element identified by choice group ID. The function returns the index of the newly inserted element within the choice group.

```
function ChoiceAppendStringImage(choiceID: integer;  
itemText:string; img:image):integer;
```

```
var choiceGroupID: integer;
    NY, LA: integer;
begin
    ShowForm;
    choiceGroupID := FormAddChoice('Where do you live?',
CF_EXCLUSIVE);
    NY := ChoiceAppendStringImage(choiceGroupID,
                                'New York',
                                LoadImage('/NY.png'));
    LA := ChoiceAppendStringImage(choiceGroupID,
                                'Los Angeles',
                                LoadImage('/LA.png'));
end.
```

MIDP1.0

None

FormAddChoiceChoiceAppendStringChoiceIsSelectedChoiceGetSelectedIndex

### 3.6.4 ChoiceGetSelectedIndex

The function returns the index of the selected choice group item, or -1 if no item is selected. For CH\_MULTIPLE choice groups, -1 is always returned, and ChoiceIsSelected should be called instead.

```
function ChoiceGetSelectedIndex(choiceID: integer):integer;
```

MIDP1.0

None

FormAddChoiceChoiceIsSelected

### 3.6.5 ChoiceIsSelected

The function returns true if the item (identified by 'itemIndex') is selected in choice group (identified by 'choiceID').

```
function ChoiceIsSelected(choiceID: integer;
itemIndex:integer):boolean;
```

```
var choiceGroupID: integer;
    NY, LA: integer;
begin
    ShowForm;
    choiceGroupID := FormAddChoice('Where do you live?',
CF_EXCLUSIVE);
    NY := ChoiceAppendStringImage(choiceGroupID,
                                'New York',
                                LoadImage('/NY.png'));
    LA := ChoiceAppendStringImage(choiceGroupID,
                                'Los Angeles',
```

```
LoadImage( '/LA.png' ));

if choiceIsSelected(choiceGroupID, NY) then
    FormAddString('New York');
else
    FormAddString('Los Angeles');
end.

MIDP1.0
None
FormAddChoice
```

### 3.6.6 ClearForm

Removes all elements and commands from the form.

```
procedure ClearForm;

var label_id, textField_id: integer;
begin
    label_id := FormAddString('Hello world');
    textField_id := FormAddTextField('Enter your name',
'Mr.Smith', 20, TF_ANY);
    ShowForm;
    Delay(2000);
    ClearForm;
    Delay(2000);
end.

MIDP1.0
None
ShowForm
```

### 3.6.7 CreateCommand

Creates the command element with the label 'label'. The label should be as short as possible. The priority of the command is set with the 'priority' parameter; the lower value means higher priority. The application uses the 'commandType' to specify the intent of this command. For example, if the application specifies that the command is of type CM\_BACK, and if the device has a standard of placing the "back" operation on a certain soft-button, the implementation can follow the style of the device by using the semantic information as a guide. 'commandType' can be any of the following:

- CM\_SCREEN - any command type
- CM\_BACK

- CM\_CANCEL
- CM\_OK
- CM\_HELP
- CM\_STOP
- CM\_EXIT
- CM\_ITEM

```
function CreateCommand(label:string;
                      commandType:integer;
                      priority:integer): command;

var exitCmd, pauseCmd: command;
begin
    exitCmd := CreateCommand('Exit', CM_EXIT, 1);
    pauseCmd := CreateCommand('Pause', CM_SCREEN, 1);
    AddCommand(exitCmd);
    AddCommand(pauseCmd);
end.
```

MIDP1.0

None

GetClickedCommandAddCommandRemoveCommand

### 3.6.8 EmptyCommand

Returns a nonclicked command. The returned command **MAY NOT** be added to the screen (otherwise the MIDlet will crash), it can only be used for comparing if something has been clicked.

```
function EmptyCommand:command;

var ok, clicked:command;
begin
    ok := CreateCommand('OK', CM_OK, 1);
    AddCommand(ok);
    repeat
        clicked := GetClickedCommand;
    until clicked <> EmptyCommand;
    if clicked = ok then halt else doSomething...
end.
```

MIDP1.0

None

CreateCommandGetClickedCommandAddCommandRemoveCommand



### 3.6.9 FormAddChoice

Adds the choice group element to the form. The function returns ID of the new choice group element. The 'choiceType' can be any of the following:

- CH\_EXCLUSIVE - exactly one item must be selected at a time
- CH\_MULTIPLE - arbitrary number of elements may be selected

```
function FormAddChoice(label:string;  
choiceType:integer):integer;
```

MIDP1.0

None

ChoiceAppendStringChoiceAppendStringImageChoiceIsSelectedChoiceGetSelected  
IndexFormRemove

### 3.6.10 FormAddGauge

Inserts a gauge element to the form. The function returns the ID of the text field. If 'isInteractive' is set to false, the user can not change the value of the gauge.

```
function FormAddGauge(label:string;  
isInteractive:boolean;  
maxValue, initialValue:integer  
) :integer;  
  
var gauge_id: integer;  
begin  
    gauge_id := FormAddGauge('Choose your age', true, 100, 18);  
    ShowForm;  
    Delay(2000);  
end.
```

MIDP1.0

None

ShowFormFormSetValueFormGetValueFormRemove

### 3.6.11 FormAddImage

Inserts an image to the form. Returns the ID of the image element.

```
function FormAddImage(i:image):integer;
```

```
var image_id: integer;
begin
    image_id := FormAddImage(LoadImage('/logo.png'));
    ShowForm;
    Delay(2000);
end.
```

MIDP1.0

None

ShowFormFormRemove

### 3.6.12 FormAddSpace

Inserts a space on the form. The space is used to separate groups of elements. Returns the ID of the space element.

```
function FormAddSpace:integer;
```

```
var label_id, space_id, textField_id: integer;
begin
    label_id := FormAddString('Hello world');
    space_id := FormAddSpace();
    textField_id := FormAddTextField('Enter your name',
                                     'Mr.Smith', 20, TF_ANY);
    ShowForm;
    Delay(2000);
end.
```

MIDP1.0

None

ShowFormFormRemove

### 3.6.13 FormAddString

Inserts a (uneditable) string label on to the form. The function returns the ID of a newly inserted element.

```
function FormAddString(s:string):integer;
```

```
var label_id: integer;
begin
    label_id := FormAddString('Hello world');
    ShowForm;
    Delay(2000);
end.
```

MIDP1.0

None  
ShowFormFormRemove

### 3.6.14 FormAddTextField

Inserts a text field to the form. The function returns the ID of the text field. The 'prompt' is the string displayed next to text field. 'defaultValue' is the text that is initially in the text field. 'maxSize' is the maximum length of the text field in number of characters. 'constraints' can be any of the following:

- TF\_ANY - text field can contain any characters
- TF\_EMAIL - only email can be entered into text field
- TF\_NUMERIC - only number can be entered into text field
- TF\_PHONENUMBER - only phonenumber can be entered into text field
- TF\_URL - only URL can be entered into the text field
- TF\_PASSWORD - the text in the field is hidden, '\*' character are displayed instead

```
function FormAddTextField(prompt, defaultValue: string;  
                           maxSize: integer;  
                           constraints: integer;  
                           r  
                           ): integer;  
  
var textField_id: integer;  
begin  
    textField_id := FormAddTextField('Enter your name',  
    'Mr.Smith', 20, TF_ANY);  
    ShowForm;  
    Delay(2000);  
end.  
  
MIDP1.0  
None  
ShowFormFormSetTextFormGetTextFormRemove
```

### 3.6.15 FormGetText

Returns the text in the text field identified by its ID. If the 'textFieldID' does not identify a text field element, an empty string is returned.

```
function FormGetText(textFieldID: integer): string;
```

MIDP1.0  
None  
FormAddText

### 3.6.16 FormGetValue

Returns the value in the gauge identified by its ID. If the 'gaugeID' does not identify a gauge element, -1 is returned.

```
function FormGetValue(gaugeID:integer):integer;
```

```
var gauge_id, value: integer;
begin
    gauge_id := FormAddGauge('Choose your age', true, 100, 18);
    ShowForm;
    value := FormGetValue(gauge_id);
    Delay(2000);
end.
```

MIDP1.0

None

FormAddGauge

### 3.6.17 FormRemove

Removes an element from the form.

```
procedure FormRemove(item:integer);
```

```
var textField:integer;
begin
    showForm;
    textField := FormAddTextField('Name', '', 20, TF_ANY);
    Delay(1000);
    FormRemove(textField);
end.
```

MIDP1.0

None

### 3.6.18 FormSetText

Sets the text of the text field identified by the text field ID.

```
procedure FormSetText(textFieldID:integer; text:string);
```

MIDP1.0  
None  
FormAddTextField

### 3.6.19 FormSetValue

Sets the value in the gauge identified by gauge ID.

```
procedure FormSetValue(gaugeID:integer; value:integer);
```

MIDP1.0  
None  
FormAddGauge

### 3.6.20 GetClickedCommand

Returns the last command that has been clicked.

```
function GetClickedCommand: command;  
  
var exitCmd, clicked: command;  
begin  
    exitCmd := CreateCommand('Exit', CM_EXIT, 1);  
    AddCommand(exitCmd);  
    repeat  
        clicked := GetClickedCommand;  
    until clicked <> EmptyCommand;  
end.
```

MIDP1.0  
None  
CreateCommandAddCommandRemoveCommandEmptyCommand

### 3.6.21 GetTextBoxString

Returns the text entered into the text box.

```
function GetTextBoxString:string;  
  
var cont : command;  
    quote : string;  
begin  
    ShowTextBox('Enter your favorite quote', 'To be or not to  
be', 200, TF_ANY);  
    cont := CreateCommand('Continue', CM_SCREEN, 1);  
    AddCommand(cont);
```

```

    repeat
        delay(100);
    until GetClickedCommand <> EmptyCommand;

    quote := GetTextBoxString;

    ...
end.

MIDP1.0
None
ShowTextBox

```

### 3.6.22 MenuAppendString

Appends a string entry to current menu. The function returns the index of the new entry within the menu.

```

function MenuAppendString(text: string): integer;

var tetris, minesweeper, snake : integer;
    play, clicked : command;
begin
    ShowMenu('Select a game', CH_IMPLICIT);

    tetris      := MenuAppendString('Tetris');
    minesweeper := MenuAppendString('Minesweeper');
    snake       := MenuAppendString('Snake');

    play := CreateCommand('Play', CM_SCREEN, 1);
    AddCommand(play);

    repeat
        delay(100);
        clicked := GetClickedCommand;
    until clicked = play;

    ShowCanvas; // show canvas and remove menu from the screen

    if MenuGetSelectedIndex = tetris then PlayTetris;
    if MenuGetSelectedIndex = minesweeper then PlayMinesweeper;
    if MenuGetSelectedIndex = snake then PlaySnake;
    ...
end.

MIDP1.0
None
ShowMenuMenuAppendStringImageMenuIsSelectedMenuGetSelectedIndex

```

### 3.6.23 MenuAppendStringImage

Appends a string entry to current menu. The entry contains an image next to the text. The function returns the index of the new entry within the menu.

```
function MenuAppendStringImage(text: string; img:image):
integer;

var tetris, minesweeper, snake : integer;
    play, clicked : command;
begin
    ShowMenu('Select a game', CH_IMPLICIT);

    tetris      := MenuAppendStringImage('Tetris',
LoadImage('/tetris.png'));
    minesweeper := MenuAppendStringImage('Minesweeper',
LoadImage('/mine.png'));
    snake       := MenuAppendStringImage('Snake',
LoadImage('/snake.png'));

    play := CreateCommand('Play', CM_SCREEN, 1);
    AddCommand(play);

    repeat
        Delay(100);
        clicked := GetClickedCommand;
    until clicked = play;

    ShowCanvas; // show canvas and remove menu from the screen

    if MenuGetSelectedIndex = tetris then PlayTetris;
    if MenuGetSelectedIndex = minesweeper then PlayMinesweeper;
    if MenuGetSelectedIndex = snake then PlaySnake;
    ...
end.
```

MIDP1.0

None

ShowMenuMenuAppendStringMenuIsSelectedMenuGetSelectedIndex

### 3.6.24 MenuGetSelectedIndex

Returns the index of the currently selected entry in the menu. Returns -1 if no entry is selected.

```
function MenuGetSelectedIndex: integer;

var tetris, minesweeper, snake : integer;
    play, clicked : command;
begin
```

```

    ShowMenu('Select a game', CH_IMPLICIT);

    tetris      := MenuAppendStringImage('Tetris',
LoadImage('/tetris.png'));
    minesweeper := MenuAppendStringImage('Minesweeper',
LoadImage('/mine.png'));
    snake       := MenuAppendStringImage('Snake',
LoadImage('/snake.png'));

    play := CreateCommand('Play', CM_SCREEN, 1);
    AddCommand(play);

    repeat
        Delay(100);
        clicked := GetClickedCommand;
    until clicked = play;

    ShowCanvas; // show canvas and remove menu from the screen

    if MenuGetSelectedIndex = tetris then PlayTetris;
    if MenuGetSelectedIndex = minesweeper then PlayMinesweeper;
    if MenuGetSelectedIndex = snake then PlaySnake;
    ...
end.

MIDP1.0
None
ShowMenuMenuAppendStringMenuAppendStringImageMenuIsSelected

```

### 3.6.25 MenuIsSelected

Returns true if the entry at 'index' within the menu is selected.

```

function MenuIsSelected(index: integer): boolean;

var tetris, minesweeper, snake : integer;
    play, clicked : command;
begin
    ShowMenu('Select a game', CH_IMPLICIT);

    tetris      := MenuAppendStringImage('Tetris',
LoadImage('/tetris.png'));
    minesweeper := MenuAppendStringImage('Minesweeper',
LoadImage('/mine.png'));
    snake       := MenuAppendStringImage('Snake',
LoadImage('/snake.png'));

    play := CreateCommand('Play', CM_SCREEN, 1);
    AddCommand(play);

    repeat

```



```
        Delay(100);
        clicked := GetClickedCommand;
    until clicked = play;

    ShowCanvas; // show canvas and remove menu from the screen

    if MenuIsSelected(tetris) then PlayTetris;
    if MenuIsSelected(minesweeper) then PlayMinesweeper;
    if MenuIsSelected(snake) then PlaySnake;
    ...
end.

MIDP1.0
None
ShowMenuMenuAppendStringMenuAppendStringImageMenuIsSelected
```

### 3.6.26 PlayAlertSound

Plays a sound associated with the current alert.

```
procedure PlayAlertSound;

var cm : command;

begin
    ShowAlert('Message',
              'New message arrived',
              LoadImage('/img1.png'),
              ALERT_INFO);

    PlayAlertSound;

    cm := CreateCommand('OK', CM_OK, 1);
    AddCommand(cm);

    repeat
        Delay(100);
    until GetClickedCommand <> EmptyCommand;

    ShowForm; // this will clear alert from the screen
    ...
end.

MIDP1.0
None
ShowAlert
```

### 3.6.27 RemoveCommand

Removes a command from the device display.

```
procedure RemoveCommand(cmd: command);
```

MIDP1.0

None

CreateCommandGetClickedCommandAddCommandEmptyCommand

### 3.6.28 SetTicker

Adds a ticker-tape effect to the form.

```
procedure SetTicker(s:string);
```

```
begin
    ShowForm;
    SetTicker('MIDlet created with MIDletPascal');
    Delay(5000);
end.
```

MIDP1.0

None

### 3.6.29 ShowAlert

Displays an alert on the screen. The screen cannot contain any other form elements (except commands) when an alert is displayed. An alert can play the sound when it is displayed. Alert type can be any of the following:

- ALERT\_INFO
- ALERT\_WARNING
- ALERT\_ERROR
- ALERT\_ALARM
- ALERT\_CONFIRMATION

```
procedure ShowAlert(title: string; message:string; img:image;
alertType:alert);
```

```
var cm : command;
begin
    showAlert('New message',
```

```
        'You have just received a message from MrSmith',
        loadImage('/img1.png'),
        ALERT_INFO);
playAlertSound;
cm := createCommand('Read', CM_OK, 1);
addCommand(cm);
repeat
    delay(100);
until getClickedCommand <> emptyCommand;
showForm; // this will clear alert from the screen
...
end.
```

MIDP1.0

None

### 3.6.30 ShowCanvas

Displays the canvas on the device screen. The device screen can show either form or canvas. The form contains labels, images, text fields and other user interface elements. The canvas, on the other hand, does not contain user interface elements; it is an area that is painted by the program. When the MIDlet is started, the canvas is shown on the screen.

```
procedure ShowCanvas;
```

```
var label_id, textField_id: integer;
begin
    label_id := FormAddString('Hello world');
    textField_id := FormAddTextField('Enter your name',
'Mr.Smith', 20, TF_ANY);
    ShowForm;
    Delay(2000);
    ShowCanvas;
    DrawText('Hello world', 0, 0);
    Repaint;
    Delay(2000);
end.
```

MIDP1.0

None

ShowFormDrawing on the screen

### 3.6.31 ShowForm

Displays the form on the device screen. The device screen can show either form or canvas. The form contains labels, images, text fields and other user interface elements. The canvas, on the other hand, does not contain user interface elements;

it is an area that is painted by the program.

```
procedure ShowForm;
```

```
var label_id, textField_id: integer;
begin
    label_id := formAddString('Hello world');
    textField_id := formAddTextField('Enter your name',
                                     'Mr.Smith', 20, TF_ANY);
    showForm;
    delay(2000);
end.
```

MIDP1.0  
None

### 3.6.32 ShowMenu

Shows a menu on the device display. No other form elements (excepts commands) can be added to the screen when a menu is displayed. 'menuType' can be any of the following:

- CH\_IMPLICIT - this is what you probably want
- CH\_EXCLUSIVE - a small radio button is displayed next to each menu entry
- CH\_MULTIPLE - you can select multiple entries

```
procedure ShowMenu(title:string; menuType:integer);
```

```
var tetris, minesweeper, snake : integer;
    play, clicked : command;
```

```
begin
    showMenu('Select a game', CH_IMPLICIT);

    tetris := menuAppendString('Tetris');
    minesweeper := menuAppendString('Minesweeper');
    snake := menuAppendString('Snake');

    play := createCommand('Play', CM_SCREEN, 1);
    addCommand(play);
    repeat
        delay(100);
        clicked := getClickedCommand;
    until clicked = play;
    showCanvas; // show canvas and remove menu from the screen

    if menuGetSelectedIndex = tetris then playTetris;
    if menuGetSelectedIndex = minesweeper then playMinesweeper;
    if menuGetSelectedIndex = snake then playSnake;
    ...
end;
```

end.

MIDP1.0

None

### 3.6.33 ShowTextBox

Shows the text box on the screen. Text box occupies the whole device display and no other form elements except commands can be added to the screen. The 'constraints' can be any of the following:

- TF\_ANY - text field can contain any characters
- TF\_EMAIL - only email can be entered into text field
- TF\_NUMERIC - only number can be entered into text field
- TF\_PHONENUMBER - only phonenumber can be entered into text field
- TF\_URL - only URL can be entered into the text field

```
procedure showTextBox(title: string;
                      initialContents: string;
                      maxSize: integer;
                      constraints: integer);

var cont : command;
    quote : string;
begin
    showTextBox('Enter message, ', 200, TF_ANY);
    cont := createCommand('Send', CM_SCREEN, 1);
    addCommand(cont);

    repeat
        delay(100);
    until getClickedCommand <> emptyCommand;

    quote := getTextBoxString;

    ...
end.
```

MIDP1.0

None

## 3.7 Record Store

### 3.7.1 AddRecordStoreEntry

The function adds 'data' into the record store 'rs'. The function returns the index inside record store at which 'data' is stored, or -1 if an error occurred.

```
function AddRecordStoreEntry(rs: recordStore; data: string):  
integer;
```

MIDP1.0

None

DeleteRecordStoreEntry

### 3.7.2 CloseRecordStore

Closes an open record store.

```
procedure CloseRecordStore(rs: recordStore);
```

MIDP1.0

None

### 3.7.3 DeleteRecordStore

Deletes the record store identified by its name. All data stored inside the record store will be lost. If the record store with the given name does not exist, nothing is deleted.

```
procedure DeleteRecordStore(name: string);
```

MIDP1.0

None

### 3.7.4 DeleteRecordStoreEntry

Deletes the entry located at index 'index' inside the 'rs' recordStore. The procedure will do nothing if nothing exists at the given index.

```
procedure DeleteRecordStoreEntry(rs: recordStore; index:  
integer);
```

MIDP1.0  
None

### 3.7.5 GetRecordStoreNextId

Returns the entry index that will be given to the next record store entry created with 'AddRecordStoreEntry' function.

```
function GetRecordStoreNextId(rs: recordStore): integer;
```

MIDP1.0  
None  
AddRecordStoreEntry

### 3.7.6 GetRecordStoreSize

Returns the number of records in the record store.

```
function GetRecordStoreSize(rs: recordStore): integer;
```

MIDP1.0  
None  
GetRecordStoreNextId

### 3.7.7 ModifyRecordStoreEntry

Modifies existing record store entry identified by 'index'. The value of the record store entry is set to 'newData'.

```
procedure GetRecordStoreSize(rs: recordStore;  
                             newData: string;  
                             index: integer);
```

MIDP1.0  
None

### 3.7.8 OpenRecordStore

Open the record store with the name 'name'. If the record store does not exist, an empty record store is created.

```
function OpenRecordStore(name: string): recordStore;
```

MIDP1.0  
None

CloseRecordStore

### 3.7.9 ReadRecordStoreEntry

Returns the data stored at index 'index' inside record store 'rs', or an empty string if nothing exists at the given index.

```
function ReadRecordStoreEntry(rs: recordStore; index: integer):  
string;
```

MIDP1.0

None



## 3.8 Http Connectivity

### 3.8.1 AddHttpBody

Appends 'data' to the http request message body. Only POST request may contain a non-empty body.

```
procedure AddHttpBody(httpConn: http; data: string);
```

MIDP1.0  
None

### 3.8.2 AddHTTPHeader

Inserts a http header into http request. For example, to add the header "Accept-encoding: gzip, none", call 'addHTTPHeader' with 'name' set to "Accept-encoding", and 'value' set to "gzip, none".

```
procedure AddHTTPHeader(httpConn: http; name, value:string);
```

MIDP1.0  
None

### 3.8.3 CloseHttp

Closes an open http connection.

```
procedure CloseHttp(httpConn: http);
```

MIDP1.0  
None

### 3.8.4 GetHTTPHeader

Returns the value of the header 'name' from the http response. For example, if the http response contains the header "Content-type: text/plain", calling 'getHTTPHeader' with 'name' set to "Content-type" will return "text/plain".

```
function GetHTTPHeader(httpConn: http; name: string): string;
```

MIDP1.0

None

### 3.8.5 GetHttpResponse

Returns the data that has been sent by the server in a response to http request.

```
function GetHttpResponse(httpConn: http):string;
```

MIDP1.0

None

### 3.8.6 IsHttpOpen

Returns true if the 'httpConn' connection is already open.

```
function IsHttpOpen(httpConn: http):boolean;
```

MIDP1.0

None

### 3.8.7 OpenHttp

Opens the 'httpConn' connection to the 'url'. The 'url' must be in form 'http://servername[:port][/file]'. Returns true on success, or false if fails.

```
function OpenHttp(httpConn: http; url: string):boolean;
```

MIDP1.0

None

### 3.8.8 SendHttpMessage

This function send the http request, waits for the response and returns http response code or -1 if an internal error occurred. The complete list of HTTP response codes can be found on the Internet (generally, 200 means OK, codes between 400-499 denote client-side errors, and codes between 500-599 denote server-side errors).

```
function SendHttpMessage(httpConn: http): integer;
```

MIDP1.0  
None

### 3.8.9 SendHttpMethod

Sets the method for the http request. The method can be any of the following:

- GET
- POST
- HEAD

```
procedure SetHttpMethod(httpConn: http; method:string);
```

MIDP1.0  
None

## 3.9 SMS Messaging

### 3.9.1 SmsIsSending

Returns true if SMS subsystem is busy sending an SMS message, or false otherwise.

```
function SmsIsSending: boolean;
```

MIDP1.0

None

SmsStartSendSmsWasSuccessfull

### 3.9.2 SmsStartSend

Starts sending an SMS message. Returns true if SMS message has been sent to the SMS subsystem and if the subsystem is not sending another message. Returns false if the message was not sent to the SMS subsystem. 'destination' address must have the form: sms://<phone-number>

SMS sending from J2ME is not supported on all devices. MIDletPascal tries to send SMS messages using Wireless Messaging API and Siemens API.

The following example illustrates simple use of SMS functions:

```
function SmsStartSend(destination: string; message:string):  
boolean;
```

```
begin
```

```
    if not SmsStartSend('sms://+5550000', 'Hello!') then Halt;
```

```
        while SmsIsSending do // wait for the message to be sent  
            Delay(100);
```

```
            if not SmsWasSuccessfull then Halt; // check if the  
message was sent  
            //successfully
```

```
end.
```

MIDP1.0

None

SmsIsSendingSmsWasSuccessfull

### 3.9.3 SmsWasSuccessful

Returns true if the last SMS message was sent successfully. This function returning false can mean any of the following:

- the mobile phone does not support sending SMS messages from J2ME MIDlet
- the SMS destination address is invalid
- the operator refused to relay an SMS message

```
function SmsWasSuccessful: boolean;
```

MIDP1.0

None

SmsStartSendSmsIsSending

## 3.10 Sound and Music

### 3.10.1 GetPlayerDuration

Returns the total duration (in milliseconds) of the music that was loaded into player.

```
function GetPlayerDuration:integer;
```

MMAPI or MIDP2.0

None

OpenPlayer

### 3.10.2 OpenPlayer

Opens a given resource file in the audio player. The resource is not played before 'startPlayer' is called. The function will return false if it did not succeed. 'mimetype' can be any of the following:

- wave files: audio/x-wav
- au files: audio/basic
- MP3 files: audio/mpeg
- MIDI files: audio/midi

Note that devices usually do not support all sound formats.

**Compatibility:** the music functions will work only on MIDP-2.0 compatible mobile phones. MIDlets with sound support will CRASH on MIDP-1.0 phones!

```
function OpenPlayer(resource:string; mimetype:string):boolean;
```

```
begin
```

```
    if not OpenPlayer('/explosion.mid', 'audio/midi') then  
Halt;  
    if not SetPlayerCount(-1) then Halt;  
    if not StartPlayer then Halt;
```

```
    Delay(5000);  
end.
```

MMAPI or MIDP2.0

None

### 3.10.3 SetPlayerCount

Sets the loop count for the player - the 'loopCount' is the number of times that the music should be played. This function should be called after 'openPlayer' and before 'startPlayer' is called. If 'loopCount' is set to -1, then the music will play forever.

```
function SetPlayerCount(loopCount:integer):boolean;
```

MMAPI or MIDP2.0

None

OpenPlayer

### 3.10.4 StartPlayer

Starts playing the audio previously loaded by 'openPlayer'. Returns false if the player cannot be started.

```
function StartPlayer:boolean;
```

```
begin
```

```
    if not OpenPlayer('/explosion.mid', 'audio/midi') then  
Halt;  
    if not SetPlayerCount(-1) then Halt;  
    if not StartPlayer then Halt;
```

```
    Delay(5000);  
end.
```

MMAPI or MIDP2.0

None

OpenPlayer

### 3.10.5 StopPlayer

Stops the music that is currently played.

```
procedure StopPlayer;
```

MMAPI or MIDP2.0

None

OpenPlayerStartPlayer

## 3.11 Resource Files

### 3.11.1 CloseResource

Closes the given resource.

```
procedure CloseResource(res: resource);
```

MIDP1.0

None

OpenResourceResourceAvailableReadByteReadLine

### 3.11.2 OpenResource

The function opens the resource file located within the application's JAR file. To add the resource file into the JAR file, select the Project->Insert resource menu.

```
function OpenResource(name: string):resource;
```

```
var res    : resource;  
    byte   : integer;  
    line   : string;  
    index  : integer;  
begin  
    res := OpenResource('/data.txt');  
  
    if (resourceAvailable(res)) then  
    begin  
        byte := ReadByte(res);  
        line := ReadLine(res);  
  
        CloseResource(res);  
    end;  
  
    ShowForm;  
    index := FormAddString('Byte is: ' + chr(byte));  
    index := FormAddString('Line is: ' + line);  
    Delay(1000);  
end.
```

MIDP1.0

None

ResourceAvailableCloseResourceReadByteReadLine

### 3.11.3 ReadByte

Reads the next byte from the given resource, or returns EOF if there is nothing to read or there was an error reading from the resource.



```
function ReadByte(res: resource):integer;
```

MIDP1.0

None

OpenResourceResourceAvailableCloseResourceReadLine

### 3.11.4 ReadLine

Reads the next line from the given resource, returns an empty string if there were no more lines to read or if there was an error reading from the stream. If there are empty lines in the resource, they will be skipped.

```
function ReadLine(res: resource):string;
```

MIDP1.0

None

OpenResourceResourceAvailableCloseResourceReadByte

### 3.11.5 ResourceAvailable

The function returns true if the given resource is available (e.g. if it was open correctly).

```
function ResourceAvailable(res: resource):boolean;
```

MIDP1.0

None

OpenResourceCloseResourceReadByteReadLine

## 3.12 Misc

### 3.12.1 Assert

This procedure test if the given condition is true. If the given condition fails (it is not true), then an assertion message is written to the standard debug output. More information about debug output can be found with the definition of debug procedure. The sample assetion message looks like this:

```
Assertion failed at: Tetris.mpsrc:162
```

```
procedure Assert(cond: boolean);
```

MIDP1.0

None

Debug

### 3.12.2 Chr

Returns the character with the given ASCII code. Behavior is undefined if 'n' is larger than 127.

```
function Chr(n: integer): char;
```

MIDP1.0

None

### 3.12.3 Debug

This procedure writes the given string to the simulator's standard output. It has no effect on the mobile device (except that the generated MIDlet is slightly larger).

```
procedure Debug(s: string);
```

MIDP1.0

None

Assert

### 3.12.4 GetProperty

This procedure is used to get the information about the Java system. Consult the Java documentation for the list of available properties.

```
function GetProperty(propertyName: string): string;
```

```
begin
    Debug(GetProperty('microedition.locale'));
end.
```

MIDP1.0

None

DebugGetWidthGetHeight

### 3.12.5 Halt

Terminates the MIDlet execution.

```
procedure Halt;
```

MIDP1.0

None

IsMidletPaused

### 3.12.6 IsMidletPaused

Returns true if the MIDlet is in the paused state, otherwise false.

When the MIDlet is started, it is not in the paused state. The MIDlet can enter paused state when, for example, the phone receives incoming call. Then the MIDlet enters paused state, the call is answered, and after the phone call the user may resume the MIDlet - after resuming the MIDlet is not in the paused state any more.

Determining if the MIDlet is in the paused state may be useful in different applications. Consider, for example, a game that must be paused when the MIDlet is paused. The following code would do the trick:

```
function IsMidletPaused: boolean;
```

```
...
repeat
    { process keypad inputs and read the timer }

    { if the MIDlet is paused, wait until it is resumed }
    while IsMidletPaused do
    begin
```

```
        Delay(100);
    end;

    until gameOver;
    ...

MIDP1.0
None
GetKeyCodeGetRelativeTimeMsHalt
```

### 3.12.7 Odd

Returns true if the given number 'n' is odd, false otherwise.

```
function Odd(n: integer): boolean;
```

MIDP1.0  
None

### 3.12.8 Ord

Returns the ASCII character code for the given character.

```
function Ord(c: char): integer;
```

MIDP1.0  
None

### 3.12.9 Random

Returns the pseudorandom number between 0 and (n-1).  
The following function will return random boolean value; the probability that the returned value is true is 75 %.

```
function Random(n: integer): integer;
```

```
function randomBoolean: boolean;
begin
    if Random(4) = 3 then
```

```
        randomBoolean := false;  
    else  
        randomBoolean := true;  
end;
```

MIDP1.0  
None  
Randomize

### 3.12.10 Randomize

Reinitialize the random number generator. The MIDlet does this initialization when it is run; however, you may wish to reinitialize the random number generator at some time.

```
procedure Randomize;
```

MIDP1.0  
None



# Index

## - A -

Abs 66  
Accessing Java from MIDletPascal 34  
Acos 66  
AddCommand 75  
AddHttpBody 95  
AddHTTPHeader 95  
AddRecordStoreEntry 92  
Alert 25, 87  
ALERT\_ALARM 88  
ALERT\_CONFIRMATION 88  
ALERT\_ERROR 88  
ALERT\_INFO 88  
ALERT\_WARNING 88  
Arc 46  
ASCII 104, 106  
Asin 66  
Assert 34, 104  
Atan 67  
Atan2 67

## - B -

Backlight 34  
Building project 13

## - C -

Canvas 25, 89  
CH\_EXCLUSIVE 79, 90  
CH\_IMPLICIT 90  
CH\_MULTIPLE 79, 90  
Character 72, 73  
Choice group 75, 76, 79  
ChoiceAppendString 75  
ChoiceAppendStringImage 75  
ChoiceGetSelectedIndex 76  
ChoiceIsSelected 76  
Chr 104  
ClearForm 77  
Clipping area 55  
CloseHttp 95  
CloseRecordStore 92  
CloseResource 102  
Color 49, 50, 51, 53, 54, 55  
Command 75, 77, 78, 83, 88

Compiling project 13  
Connectivity 32  
Copy 72  
Cos 67  
CreateCommand 77  
Current time 61, 63

## - D -

Day 62, 65  
Debug 34, 104  
Debugging 34  
Delay 61  
DeleteRecordStore 92  
DeleteRecordStoreEntry 92  
Display 54  
DrawArc 46  
DrawEllipse 46  
DrawImage 46  
Drawing 19  
DrawLine 47  
DrawRect 47  
DrawRoundRect 47  
DrawText 48

## - E -

Ellipse 46, 48  
EmptyCommand 78  
Emulators 13  
Exp 67  
Extending MIDletPascal 34

## - F -

FillEllipse 48  
FillRect 49  
FillRoundRect 49  
Flash memory 30, 43  
FlashBacklight 34  
Font 56  
FONT\_FACE\_MONOSPACE 56  
FONT\_FACE\_PROPORTIONAL 56  
FONT\_FACE\_SYSTEM 56  
FONT\_SIZE\_LARGE 56  
FONT\_SIZE\_MEDIUM 56  
FONT\_SIZE\_SMALL 56  
FONT\_STYLE\_BOLD 56  
FONT\_STYLE\_ITALIC 56  
FONT\_STYLE\_PLAIN 56  
FONT\_STYLE\_UNDERLINE 56

Form 25, 89  
FormAddChoice 79  
FormAddGauge 79  
FormAddImage 79  
FormAddSpace 80  
FormAddString 80  
FormAddTextField 81  
FormGetText 81  
FormGetValue 82  
FormRemove 82  
FormSetText 82  
FormSetValue 83  
Frac 68

## - G -

GA\_DOWN 59  
GA\_FIRE 59  
GA\_GAMEA 59  
GA\_GAMEB 59  
GA\_GAMEC 59  
GA\_GAMED 59  
GA\_LEFT 59  
GA\_NONE 59  
GA\_RIGHT 59  
GA\_UP 59  
Gauge 25, 79, 82, 83  
GET 97  
GetChar 72  
GetClickedCommand 83  
GetColorBlue 49  
GetColorGreen 50  
GetColorRed 50  
GetColorsNum 51  
GetCurrentTime 61  
GetDay 62  
GetHeight 51  
GetHour 62  
GetHTTPHeader 95  
GetHttpResponse 96  
GetImageHeight 51  
GetImageWidth 52  
GetKeyClicked 58  
GetKeyPressed 58  
GetMinute 62  
GetMonth 63  
GetPlayerDuration 100  
GetProperty 105  
GetRecordStoreSize 93  
GetRelativeTimeMs 63  
GetSecond 64

GetStringHeight 52  
GetStringWidth 52  
GetTextBoxString 83  
GetWeekDay 65  
GetWidth 53  
GetYear 65  
GetYearDay 65  
GPRS 32

## - H -

Halt 105  
HEAD 97  
Height 51, 52  
Hello world 19  
Hour 62  
HTTP 32  
Http header 95

## - I -

IDE 13  
Image 19, 46, 51, 52, 53, 79  
Implementation 19  
Initialization 19  
Integer 72, 74  
IntegerToString 72  
Integrated development environment 13  
Interface 19  
Internet 32  
IsColorDisplay 53  
IsHttpOpen 96  
IsMidletPaused 105

## - J -

JAR archive 34, 53  
Java 34

## - K -

KE\_KEY0 58  
KE\_KEY1 58  
KE\_KEY2 58  
KE\_KEY3 58  
KE\_KEY4 58  
KE\_KEY5 58  
KE\_KEY6 58  
KE\_KEY7 58  
KE\_KEY8 58



KE\_KEY9 58  
KE\_NONE 58  
KE\_POUND 58  
KE\_STAR 58  
Key 58, 59  
Keypad 19  
KeyToAction 59

## - L -

Label 25, 80  
Length 73  
Library units 34  
Line 47  
LoadImage 53  
Locase 73  
Log 68  
Log10 68  
Lowercase 73

## - M -

Menu 25, 84, 85, 86  
MenuAppendString 84  
MenuAppendStringImage 85  
MenuGetSelectedIndex 85  
MenuIsSelected 86  
Messaging 32  
MIDP 2.0 13  
Minute 62  
Month 63

## - N -

Networking 32

## - O -

Odd 106  
OpenHttp 96  
OpenPlayer 100  
OpenRecordStore 93  
OpenResource 102  
Optimizing code 43  
Ord 106

## - P -

Performance 43  
PlayAlertSound 87

Plot 54  
Pos 73  
Position 73  
POST 95, 97  
Pow 69  
Project 13

## - R -

Rabs 69  
Random 106  
Randomize 107  
ReadByte 102  
ReadLine 103  
ReadRecordStoreEntry 94  
Real 74  
Record store 30, 92, 93, 94  
Rectangle 47, 49  
RemoveCommand 88  
Repaint 54  
ResourceAvailable 103  
Resources 13, 34  
Running MIDlet 13

## - S -

Second 64  
SendHttpMessage 96  
SendHttpMethod 97  
SetChar 73  
SetClip 55  
SetColor 55  
SetDefaultFont 56  
SetFont 56  
SetPlayerCount 101  
SetTicker 88  
ShowAlert 88  
ShowCanvas 89  
ShowForm 89  
ShowMenu 90  
ShowTextBox 91  
Sin 69  
SMS 32, 98, 99  
SmsIsSending 98  
SmsStartSend 98  
SmsWasSuccessful 99  
Sound 87  
Sqr 69  
Sqrt 70  
StartPlayer 101  
StopPlayer 101

String 72, 73, 74  
String label 25, 80  
StringToInteger 74  
StringToReal 74

## - Y -

Year 65

## - T -

Tan 70  
TCP 32  
Text 48, 52  
Text box 83  
Text field 81, 82  
Text-box 25  
Text-field 25  
TF\_ANY 81, 91  
TF\_EMAIL 81, 91  
TF\_NUMERIC 81, 91  
TF\_PASSWORD 81  
TF\_PHONENUMBER 81, 91  
TF\_URL 81, 91  
Ticker-tape 25, 88  
Time 61, 62, 63, 64, 65  
ToDegrees 70  
ToRadians 71  
Trunc 71  
Tutorial 12

## - U -

UDP 32  
Unit 34  
Units 19  
Ucase 74  
Uppercase 74  
User interfaces 25  
Uses 19, 34

## - V -

Vibrate 34

## - W -

Wait 61  
WAP 32  
Width 52, 53  
Writeln 19