Aptilis Manual

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Aptilis Manual

Thibault Jamme and Maximilian Schöfmann

This file was generated on Sunday, 11th of July 2004
The latest version is available at www.aptilis.com/documentation.html

What is Aptilis?
It is a language that allows you to create interactive web sites, like shops, booking kiosks, etc. It uses what is called the Common Gateway Interface (CGI), which is an industry standard.

Why should I use Aptilis?
Because it takes a day to learn it about 2 to become productive with it. If you can do Basic, you can do Aptilis. In your company more people can do more things, more quickly. If you're a web studio, you may be using a contractor to do your CGI programming in C. Aptilis allows you to do that yourself.
The help is available online and offline in many different file formats
Aptilis is also a nice way to learn programming.
Aptilis allows you to do more, it saves you money, and it's fun to use.

Aptilis simplifies my life. How?
Aptilis has been designed for the web, so unlike other languages, you don't have to use complicated algorithms to interface with the internet.
For example the content of a Web based form is passed to you through already defined variable.
Another brilliant example is that sending an e−mail takes only one command.
Aptilis has no complicated concepts such as objects, pointers, char arrays, etc.

Aptilis can help me keep my job, You're kidding, right?
No, really, Aptilis can help you keep your job!
One thing that often happens is that your company or the ISP the company you're working for uses, may change their webserver, for exemple from a Unix box to an NT based one, and a some time later back again... It happens all the time. Since Aptilis works exactly the same under these two platforms, your job is safe, no need to hire someone with different skills.

How does Aptilis make me more productive?
It's very high level, in other words, an Aptilis command does a lot of technical things that you don't have to worry about. For example 'sendmail'.
There is no compile stage which makes things easier to test. An error in your code won't cause a crash of your whole server as can happen in C.
There are two versions of Aptilis, a Windows 9x/NT/2k/XP one and a Unix one. So you can try your scripts on your own machine and once everything works, you can upload to the real server be it Windows or Unix. You might have to change a couple of paths, but that's all there is to it. You can work from home, or built useful applications for your intranet very quickly. Although it's not its main job, Aptilis can also help for some administrative tasks.
You get an executable for your platform, not some code you have to compile, as is often the (frustrating) case with Unix programs.

What are the main advantages of Aptilis compared to other languages?
To C/C++ and Java, Aptilis is so much easier to master!
To PERL Aptilis is a lot more legible.
Even if you're committed to C, you will still find aptilis extremely useful to create prototypes or demonstrators of your web applications.

Any disadvantages?
Aptilis is a language that works in its own specific way, so it has to have some challenges it solves better than others. Aptilis cannot address all programming needs, but it does address web related ones better.
Since aptilis is not compiled, it can be rightly argued that an aptilis program is slower than a C program. Aptilis puts convenience before performance, because the hardware has improved so much. It is less of a problem to use interpreted languages nowadays. As a reminder, Visual Basic, Perl and Java are also interpreted languages in most cases.

Is it well supported? Is it reliable?
The author is committed to fully support the product and is contactable via e−mail. He has been using Aptilis for many years within his own company and it is now very stable.
Until the beginning of 2002 a free hosting service was available and hundreds of users were using Aptilis on a daily basis. Because of that, a few bugs that could not have been found otherwise have been eradicated.
Since Aptilis has become a open source project now, improvements and bugfixes will happen even faster.

Where does the name Aptilis come from?
Originally, it was meant to be called 'abilis', after 'habilis', the first man who was able to use tools. But since abilis was taken, 'ab' has been replaced by the more latin stem 'apt', 'being able to'.

Topics

- Features
- FAQ
- For beginners
- What are operators?
- What are variables?
- File names
- How to do Web Forms
- What are Environment Variables?
- Advanced topics

Features

Topics

- Familiar, unabreviated built−in functions called 'predefined subs'
- Simple variable model, no declarations
- Easy, sorry, incomparably easy interfacing with web forms
- Web pages can be loaded within a programm, e−mail messages can be sent and received programmatically
- Templates, for minimum coding
- Revolutionary 'fillForm' command that understands HTML and allows you separate the HTML from your code.
- Graphics generation
- True type fonts in graphics
- The code is readable!
- XML
- Sockets

FAQ

Topics

What is Aptilis
Aptilis is a programing language targeted at web professionals. Because it is very simple − yet powerful − it is good for beginners too. It resembles Good'ol Basic.

What can I do with Aptilis?
Loads. Check the examples:
Web counter, feed−back−form, web based newsgroup, guest book, wap applications, on−line & on−the−fly−graphics, etc. The rest is up to you!

What is CGI?
Common Gateway Interface. It is a standard protocol (a way of doing things) that allows you to run programs behind a Web server. Instead of serving static (dumb) web pages, you can serve the output of a program to the user. The program can choose what to display depending on the circumstances. For example "Yes, your purchase is accepted" or "No, your credit card is dodgy" whereas a web page is always the same.

But I thought CGI was a language!
No, it's not. It's a protocol that most mordern programing languages can understand and follow, but none can do it as easily as Aptilis. For example, Aptilis gives you form fields from a web form directly into variables. No need for extra libraries or complicated things like that.

For beginners

Topics

1. What is a programming language?
2. Who's playing? A threesome!
3. What programms look like
4. What are variables?
5. What are Subs?
6. What's a '\' character and what do I do with it? Character strings!

1. What is a programming language?
A programming language is a language that computers understand.
Humans can use this language to order computers to do whatever they want, computers are slaves and that's politically correct!
I call this absolute freedom 'Computer Plasticity'. Indeed, as you can turn a ball of plasticine into anything you want, so can you with a computer.
However, computers are –so far, quite stupid. You can't use every day's language to ask your computer to do something. That's why most of the languages they use are simplifications of our human languages.
The main thing in computer languages is that they must be very precise. A computer can not understand the sentance 'Put that there'. That's too subtle.
Programms are stored in files.

2. A threesome
Sorry if you expected something more, ermmmm, more adult here, but the threesome I'm talking about is this one, sorry...

- The programmer
- The computer
- The user

The programmer writes programms.
The computer runs the programms.
The user uses the programmes.
Of the three, guess whose part is the most interesting?
Yes it's the programmer's. The computer has no notion of what is pleasurable and what is not anyway! But the programmer must first write programmes, then he has to imagine he's the computer:
'Ok: I have this variable that contains that, this loop that does that, and this sub that scrambles everything... Damn!'
Then the programmer must imagine he/she's wearing the user's shoes in order to make the user's job easy. Ok, that's the part that's the less frequently played, unfortunately, that might be the most important.
Programming can be frustrating at times, but it can also be utterly rewarding, especially when, as is

the case with Aptilis, it allows you to do spectacular things with very little sweat. Think about on−line purchase orders, graphics on the fly, games!! The world is yours!!

3. What programms look like

Aptilis programms, like most programms, are text files. In Windows, that means you can simply use the notepad to write a programm. In Unix, you'll have the less friendly (but they say, more powerful once you know how to use it...) vi.
Here's an example of a programm:

```
sub main

        print("Hello World!")

end sub
```

If you want to try it, save this file as "test" with the notepad (test.txt under Unix) in the directory where you put 'aptilis.exe', open a DOS box, go to the directory where aptilis.exe is and at the command prompt, type:

```
aptilis test.txt
```

Unix users should type ./aptilis.exe test.txt
What we've done was to write a programme, save it and run it.

4. Variables

Variables are boxes in which you put values, like a number.
You have probably heard about pi, that 'greek' number that knows how to compute all things circular. Well, you don't want to type '3.141592654' all the time. First, because you don't want to go into a math encyclopedia every two lines of code and second, you want something shorter, more readable. Let's see how to create a variable:

```
pi = 3.141592654
```

In aptilis, that means: put the value 3.141592654 into the variable 'pi'. I hear someone ask: 'No declarations, then?'
Nope.
If you don't know what a declaration is don't worry, you don't need to, unless you want to use Java or C.

5. Subs

Subs are pieces of programms, where you usually do one thing. By partitioning your programms in subs, you'll make your task easier when it comes to find bugs (errors). In all cases, you need at least one sub in Aptilis:
the **main** sub.

If there is no sub called main, then Aptilis will be lost and won't be able to run your programme. The sub main is an agreed meeting point between you and Aptilis.
Subs can also be seen as little factories: You feed them with raw materials (the parameters) and they return a finished product (the return value).
This sub is fed the radius of a circle and returns its circumference:

```
sub circumference(r)

        return 2 * 3.141592654 * r

end sub
```

There are two kinds of subs: predefined subs which have already been written for you, like 'print', 'saveFile', etc... and the ones you will write, which are simply called 'subs'.

6. The '\' character, and how to put a '' in a string, etc.
You put a string into a variable as follows:

```
name = "John Doe" $
```

Now how could you insert a double quote? Because double quotes mark the begining and the end of the string, if you inserted one or more of them within the string, that would first cut it and then confuse the Aptilis parser and you would get an error. The trick is to say: 'Hey! Attention please! There's gonna be a *special* character!!". You do that by typing a '\', in other words, a back−slash. For my fellow dyslexic programmers, that's the one on the left of the keyboard, not far from the 'Caps Lock' key. Here's what it's looking like:

```
name = "John \"Duck Face\" Doe" $
```

Now, if the back slash is reserved to indicate special characters, how do you type the backslash itself?? Here's the solution:

```
path = "c:\\directory" $
```

You just type two of them.

Here are all the special characters Aptilis knows:
\n Line Feed (print to next line)
\r Carriage return, might be needed for DOS files before a '\n'
\t Tab character
\" double quotes
\\ backslash
\0 ASCII 0
Aptilis also understands the '\xHH' notation where HH is a hexadecimal number, from 0 to 255, so that you can get the whole ASCII set.

What are operators?

Topics

1. Generalities
The operators are the characters or signs you use in expressions like *2+3* or *2*a + b*, etc... to carry out different operations.

Each operator has a specific priority, for example multiplications take precedence over additions. Expressions are evaluated from left to right.

The dollar sign '$' must be placed at the end of an expression to indicate that the the expression must be evaluated in a string context.
Not like: ~~if a$ = "hello" and b$ = "world"~~
but like: if a = "hello" and b = "world"$
If you need both contexts in the same expression, the way to do it is to use parentheses:
if (a = "hello"$) and c = 7

2. Math operators

Priority *1 = done first, 8 = done last*	**Operator**	**Use**
1	^	Exponent. **p** = 2 ^ 3, **p** contains 8
2	*	Multiplication. **m** = 4 * 3, **m** contains 12
2	/	Division. **d** = 10 / 3, **d** contains 3.3333
2	%	Modulo, that is the remainder of a division of an integer by another integer. If you use floating point values, they will first be transformed into integers. **m** = 10 % 3, **m** contains 1 
3	+	Addition

		a = 10 + 4, **a** contains 14
3	–	Substraction. **s** = 5 – 3, **s** contains 2
4	&	Binary and. **b** = 3 &2, **b** contains 2
4	|	Binary or. **o**= 5 | 3, **o** contains 7
6	<	'Less than'. **l** = 5 < 3, **l** contains 0 **l** = 1 < 2, **l** contains 1 This operator is mainly used in tests, like in: 'if a < b'
6	>	'Greater than'. **g** = 5 > 3, **g** contains 1 **g** = 1 > 2, **g** contains 0 This operator is mainly used in tests, like in: 'if a > b'
6	<=	'Less than or equal to'. **s** = 5 <= 3, **s** contains 0 **s** = 1 <= 2, **s** contains 1 This operator is mainly used in tests, like in: 'if a <= b'
6	>=	'Greater than or equal to'. **b** = 5 > = 3, **b** contains 1 **b** = 1 > = 2, **e** contains 0 This operator is mainly used in tests, like in: 'if a >= b'
7	!= **Or** <>	Different of. **d** = 1 != 2, **d** contains 1 **d** = 1 != 1, **d** contains 0 This operator is mainly used in tests, like in: 'if a != b'
7	=	Equality. This operator, which is the same as the assignment one, is mainly used in tests, like in: 'if a = b'
8	and	logical and. Gives 1 if the two operands are not 0. **A** = 5 and 2, **A** contains 1 **A** = 5 and 0, **A** contains 0 This operator is mainly used in tests, 'if a > 2 **and** a < 10' meaning 'if a is between 2 and 10 excluded'
8	or	logical or. Gives 1 if either one the two operands is not 0. **R** = 5 and 3, **R** contains 1 **R** = 5 and 0, **R** contains 1 **R** = 0 and 0, **R** contains 0 This operator is mainly used in tests like in: 'if a < 2 **or** b < 2'.

3. String operators

In a string context, some of the operators mentionned above don't apply. Others are used in a different way.
String operators that do comparisons return the string '1' when the comparison is successful, otherwise they return an empty string.

Priority ***3 = done first,*** ***8 = done last***	**Operator**	**Use**
3	+	Concatenation a = "hello" $ b = "world" $ c = a + " " + b $, c$ contains "hello world"
6	<	'Less than'. Comparison on the length of the strings
6	>	'Greater than'. Comparison on the length of the strings
6	<=	'Less than or equal to'. Comparison on the length of the strings
6	>=	'Greater than or equal to'. Comparison on the length of the strings
7	!= **Or** <>	Different of. This operator compares the two strings letter by letter and is case–sensitive.
7	=	Equality. This operator compares the two strings letter by letter and is case–sensitive.
8	and	logical and. "1" is returned if both strings are not empty.
8	or	logical or. "1" is returned if either one of the two string is not empty.

What are variables?

Topics

1. Generalities
2. Variable names
3. Types of variables
4. Arrays
5. Scope

1. Generalities
Variables in Aptilis, like in any other languages, are 'boxes' where values can be placed. One of the main advantages is to save some typing. It is easier to use 'pi' than having to type '3.14159265' all the time.
This is how you fill a variable with a value:

```
var = 16
```

And you can use the variable for its value at any time:

```
print(16)
```

is equivalent to:

```
print(var)
```

You don't have to declare variables, Aptilis reserves some space for them as they're created, i.e. when you first mention them.

2. Variable names
Variable names can be made up of letters and numbers but have to start with a letter. Names can be of any length. (but keep to a manageable wingspan...)
Meaningful variable names help a lot when it comes to maintenance.
Valid names: a, b, ab, a1, rate, rate5, etc...
Incorrect names: ~~1~~, ~~1a~~, ~~tax etc...~~
~~What applies to variable names also applies to sub names.~~

Variable names are case sensitive. 'A' is not the same as 'a'.

3. Types of variables
In aptilis, there is not really a problem of variable types. All variable are stored as text, and their numeric value is calculated when necessary. Sometimes it has to be indicated in which context you want an instruction to handle a variable. By default, the context is numeric. If you want to work in a string context, you have to use the dollar sign.

Example: a = "Hello" $
If you had forgotten the dollar sign, Aptilis would have tried to evaluate the string "Hello" as a number and would have filled 'a' with the value '0.0000'.
a = "hello" $
print(a) produces '0.0000'
print(a$) produces 'hello'

In fact, a$ and a are indeed the same variable unlike in some other close languages. Note that the dollar sign is added at the end of any expression as in:
print(left("hello", 2)$)
to get 'he' and not '0.0000'.

4. Arrays
In aptilis, all variables are arrays, even if you don't realize it!
For example to create an array with the first 10 multiples of 3 you can do:

```
for i=1 to 10
  m3[i] = i * 3
end for
```

Note that 'var[0]' is another way to write 'var'. That is you need not use the notation [0] to access the first element of an array.
You do not need to declare arrays prior to using them. Arrays grow as needed when you use them.
Arrays are used to retrieve values of 'MULTIPLE' select boxes in HTML. If you have such a box like in:

```
<SELECT name=choice MULTIPLE>
<OPTION>Pizza
<OPTION>Lasagne
<OPTION>Pasta
<OPTION>Chocolate Ice Cream
</SELECT>
```

The options selected by the user are returned in the choice[] array. You can check how many there are with the line:

```
n = getArraySize(choice[])
```

String values of array elements are assigned and accessed in the following fashion:

```
a[2] = "Greetings!" $
print(a[2]$)
```

5. Scope
Variables in Aptilis may be local or global. But as nothing can happen outside of a sub, even global variables need to be assigned from within a sub.

Being global means that you retain your value across subs. A global variable given the value '24' will retain that value in all subs. Local variable cease to exist outside the sub they appear in.
A corollary to that is that you can have a variable with the same name in different subs without interference, but a global variable such as '_a' is the same for every one.
Global variables need an underscore before their name.
_taxPercent is global
revenue is local.

File names

Topics

1. Introduction

When using file related functions, such as LoadFile, SaveFile, LoadDatabase, etc. Of course you can use the normal file name conventions of the platform you're developing on, but aptilis also understands 'URLs' in order to somehow bridge the Windows (9x/NT) / Unix gap.

To write a file name you can hence use the notation:
file://*path*
where *path* is the path to your file.
On windows platforms, you need to indicate disks as follows:
/disk_letter|
and you also need to replace back slashes by forward slashes.
The vertical bar is know as the 'pipe' character, its ASCII code is 124
The examples will help you decode all that!
Note that functions such as GetCurrentDirectory return a file or path name in the format used on your platform.

2. Examples

Normal ***As you would normally write them***	**URL** ***As you can also write them***
1. Windows filenames	
myFile.txt *In the current directory, wherever that might be (relative path)*	file://myFile.txt
\someDocs\myFile.txt *From the root of the current disk*	file:///someDocs/myFile.txt
D:\someDocs\myFile.txt *complete or 'absolute' path*	file:///D|/someDocs/myFile.txt
2. Unix filenames	
myFile.txt *In the current directory, wherever that might be (relative path)*	file://myFile.txt
/someDocs/myFile.txt *complete or 'absolute' path, all the way back to the root*	file:///someDocs/myFile.txt

3. URL types supported

All file related functions support the **file:**// type.
In addition, LoadFile supports the **http:**//, **https:**// and **ftp:**// types, and saveFile understands both the **file:**// and **ftp:**// protocols, allowing you to load web pages directly into a variable! Note that through the http:// protocol the entire web page is returned together with its server generated header. Look past the occurence of "\r\n\r\n" for actual content.
Of course, your computer needs to be connected to the Internet in order to use the http:// filetype if you want to retrieve files stored on a web server. You can also retrieve a file from an Intranet based server you're connected to through a LAN.

4. A sub to construct a URL compliant file name

```
sub main

        fn = getCurrentDirectory()
        url = toUrl(fn) $
        print("Original: ", fn$, "\n")
        print("URL:      ", url$, "\n")

end main


sub toUrl(fileName)

        if mid(fileName$, 2, 1) = ":" $
                es = "/" $
        end if

        fileName = replace(fileName$, ":", "|") $
        return "file://" + es + replace(fileName$, "\\", "/") $

end toUrl
```

Result:

```
Original: C:\aptilis
URL:      file:///C|/aptilis
```

How to do Web Forms

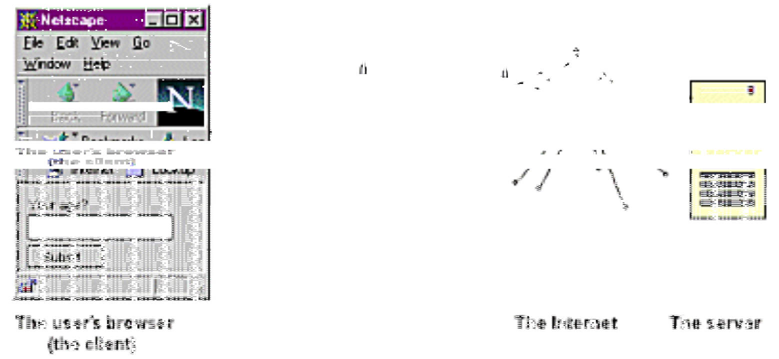
Topics

1. Introduction
Aptilis is a godsend (no, really) for the handling of Web forms.
Although aptilis follows the CGI model, you don't have to worry about the nitty gritty details!
This is a typical form (that does nothing):

Form Example:
Your email address:
Your comments:

Now the smart part is for you to write a programme which we will call a 'script' (for historical reasons).
This programme will be sent the content of the form in a certain way so that you can do something with it. Fortunately, Aptilis makes it incredibly easy for you to do such scripts.

2. How It Happens



1. The user requests a page, that happens to contain a form.
2. The server sends the page.

3. The user fills the form and clicks the submit button: the client sends the form and its collected data.
4. The sever runs the script and sends the output of the script to the user.
5. The user is happy. In most cases.

3. Form Fields
In the example above, the fields are the two boxes where you can enter your e−mail address and some comments.
If you go into the source of a form, you will notice that all fields have something in common: They have a name. This name is very important as you will use it to get the value of the different fields.
Now, one of the really nice tricks Aptilis has got up its sleeve is that you needn't write a single line of programme to retrieve the value of the fields: they have been stored in variables for you!!
Example:
If in your **form**, you have the following field:

```
<INPUT type="text" name="email" size=20>
```

all you need to do to, say, print the e−mail address given, is:

```
sub main

  // first a little magic line!!
  print("Content-type: text/html\n\n")

  print(email$)

end sub
```

Note that you shall not use dollar signs in field names, that is in the HTML part.

4. Calling an Aptilis script from a form
Calling an aptilis script form a form is simple, all you need to do is follow this guide:

- **Create the form**
  For example with the Notepad or any other text utility you're used to.

- **call the aptilis interpreter**
  You do that in the **FORM** tag of your form:

  ```
  <FORM method="http://www.myserver.com/cgi-bin/aptilis.exe" method="POST">
  ```

  Of course, you have to adapt the example given to suit your needs.
  You can also use the **GET** method, but **POST** is safer.
  If you do not know the path information for your server, ask you system admin.

- **Indicate which script to use**
  Just after the FORM tag, add the following line:

```
<INPUT type="hidden" name="file" value="myscript.e.txt">
```

  But replace 'myscript.e.txt' with the relevant value, for example:
  NT/Win95 server: **c:\webshare\aptilis-scripts\poll.e.txt**
  Unix server: **/home/httpd/aptilis-scripts/poll.e.txt**
  for a script called 'poll.e.txt'. As you have noted, the full path is needed in any case.
  If you do not know the path information for your server, ask your system admin. You can also find it when you use your FTP programme to upload files to your script directory

- **Write the script**
  ...and put it exactly where you said you would in the hidden 'file' field of your HTML form.

Note to Unix Users.
Making the script 'Executable' allows you to call the script directly into the 'action' parameter of the <FORM...> tag. In this case you don't need the hidden 'file' field, provided your system understands another magic line convention: '#!path_to_interpreter'. If you know what I'm talking about, you know where to put it.

5. Replying to a form
Your script will be called when the **user** clicks the 'submit' button. The script will be run **on the server**, NOT on the user's machine.
After the submit button has been clicked, the user's HTML is cleared. It is now up to you to re-create a new Web page for the user.
How do you do that?
Simple! You just use the **print** predefined sub to do the page:

```
sub main

  // Magic line
  print("Content-type: text/html\n\n")

  print("<HTML><BODY>\n")
  print("Hello World!!\n")
  print("</BODY></HTML>")

end sub
```

and that's all!! Of course, this was the 'Basics'. It's up to you to use the databse, direct mail (sendmail), and mathematic functions, etc... to do something a bit more useful!

6. The magic line
You have already entcountered the magic line in such code fragments:

```
// Magic line
print("Content-type: text/html\n\n")
```

(The first line starting with '//' is just a comment and does nothing)
The magic line is absolutely necessary and its role is to tell the browser (Netscape, Mosaic, Internet Explorer, etc...) what is going to be sent to it. Other magic lines are:

```
// the text will contain no HTML tags
print("Content-type: text/plain\n\n")

// A picture will be returned.
print("Content-type: image/gif\n\n")
```

The different types are called 'MIME' types and, **That's a tip** you can view some of them in Netscape, if you select the 'Options' menu, then choose 'General preferences' and finally click the 'Helpers' thumbnail.
Note that all our magic lines end with two '\n', which mean 'two carriage return', and that's also absolutely necessary for the browser to know when to start to display something.
Another tip: The magic line is actually part of the header, which can contain several other things, like a frame target for a frame–enabled browser (which worked only for Netscape until Version 4.x):

```
print("target: second_window\n")
print("Content-type: text/plain\n\n")
```

What are Environment Variables?

Topics

The environment variables are available to aptilis in a key−based array called **_ENV[ ]**. If do not understand the notion of key based arrays, just have a look at the examples below, and in particular how to retrieve a single value. You don't need much more theory to retrieve key based array elements such as the variables passed to your scripts by the webserver.

notes: I saved the example scripts in a directory called:
c:\aptilis and I called them with the URLs
http://192.1.1.16/cgi−bin/aptilis.exe?file=c:\aptilis\env1.e.txt and
http://192.1.1.16/cgi−bin/aptilis.exe?file=c:\aptilis\env2.e.txt respectively.

Example 1: Retrieving one value (script: env1.e.txt)

```
sub main

        print("Content-type: text/plain\n\n")
        print(_ENV["HTTP_HOST"]$, "\n")

end main
```

Result:

```
192.1.1.16
```

That's the IP number of my home machine

Example 2: Retrieving all available values (script: env2.e.txt)

```
sub main

        print("Content-type: text/plain\n\n")

        n = getArraySize(_ENV[])
        for i=1 to n

                k = getNextKey(_ENV[]) $
                print(k$, "=", _ENV[k$]$, "\n")

        end for

end main
```

Result:

```
CGI_STDERR=cgierr.log
QUERY_STRING=file=c:\aptilis\env2.e.txt
REMOTE_ADDR=192.1.1.16
REMOTE_HOST=192.1.1.16
REMOTE_USER=-
CONTENT_LENGTH=0
CONTENT_TYPE=
SCRIPT_NAME=/cgi-bin/aptilis.exe
```

```
SCRIPT_PATH=cgi-bin
REQUEST_METHOD=GET
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.0
SERVER_PORT=80
SERVER_SOFTWARE=Xitami
HTTP_CONTENT_LENGTH=0
HTTP_CONNECTION=Keep-Alive
HTTP_USER_AGENT=Mozilla/4.06 [en] (WinNT; I ;Nav)
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
HTTP_ACCEPT_ENCODING=gzip
HTTP_ACCEPT_LANGUAGE=en
HTTP_ACCEPT_CHARSET=iso-8859-1,*,utf-8
```

(I have removed some entries for the sake of clarity.)

Advanced topics

Importing other Aptilis scripts

1. Introduction

Imports allow you to re−use code and to keep your scripts concise.
The binary distributions of Aptilis do already include a small library of scripts, which you can use in your programs.
Aptilis has adopted the main characteristics of the package model from Java to avoid collisions between the scripts of different people. See Import conventions.

2. How to use imports

- Imports must be situated at the beginning of a program, before sub definitions.

- the syntax of the import statement is:
  import dir.subdir.subsubdir.aptilisfile [as synonym]
  Specifying a synonym is optional.

- Aptilis tries to follow the java conventions for packets:
  Directories must be separated by '.' not:
  / *or* \

- If the file is CurrencyUtils.e.txt then aptilisfile above must be:
  CurrencyUtils *not* CurrencyUtils.e.txt

- If aptilisfile is MyUtils, Aptilis will look for the following files, in this order:
  MyUtils.e.txt
  MyUtils.apt
  MyUtils.aptilis

- Aptilis will look for the file to import from the following directories in this order:
  1. Directory pointed to by the environment variable APTILISPATH
  2. Directory where the script is.
  3. Directory pointed to by the environment variable CLASSPATH (the Java one).
  4. The current directory − this may be where the interpreter (aptilis.exe) is but not necessarily. Use getCurrentDirectory() to find out where you are!

- Folders containing Aptilis files to be imported, and these Aptilis files may ONLY contain the following charater ranges in their names: a−z, A−Z, 0−9
  In other words the following characters are explicitely forbidden: _ − . / \
- File names are case sensitive under Unix, but not under windows. However in any case, RESPECT case.
- Synonyms follow variable name conventions.
  They may only contain letters, numbers, underscores, and have to start with a letter.
- Paths in the APTILISPATH follow the same rules as for java's CLASSPATH.
  - Paths are separated by ; under windows : under unix.
  - Directories in paths are separated by \ under windows and / under unix, etc.
- Directories (that are likely to be files) whose names end in .zip or .jar or .aar (for possible future aptilis archives) are ignored.
- Directories are case sensitive under Unix but NOT under Windows.

3. Import conventions:

Conventions are necessary to ensure that your own "library" doesn't come into conflict with the one of someone else.

- Packages should all have their directory names in lower case
- Filenames and synonyms should start with a capital
- Subs should start with a capital (auxiliary subs may be in lower case)
- Initialisations (global vars) should be done in an "Init" sub.
- Aptilis standard packages are available from apt.
- Any one else's packages should be their domain names reversed.
  CNN would be:
  com.cnn (...)
  SourceForge would be:
  net.sourceforge (...)
  You will need a directory structure like this: "./com/myserver/mypackages"

4. Examples

These examples use the standard Aptilis library.

Example 1:

```
import apt.template.Html as Html

sub main()

    doctype = Html.Get("doctype_4_01_strict") $

    title   = "Welcome to my Homepage" $
    body    = "<h1>Hi there!</h1>" $
    head    = "<meta name=\"author\" content=\"John Doe\">" $

    print("Content type: text/html\n\n")

    print(stuff(Html.Get("skeleton")$)$)

end main
```

Result:

```
Content type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
        <title>Welcome to my Homepage</title>
<meta name="author" content="John Doe">
</head>
<body >
<h1>Hi there!</h1>
</body>
</html>
```

Example 2:

```
import apt.http.Cookie

sub main()

    cookiedata["ip"] = _ENV["REMOTE_ADDR"] $
    cookiedata["browser"] = _ENV["HTTP_USER_AGENT"] $

    apt.http.Cookie.Set(cookiedata[],"")

    print("Content-type: text/plain\n\n")

    oldcookie[] = apt.http.Cookie.Get()

    if cookiedata["browser"] != oldcookie["browser"] $
        print("Hey! You changed your browser!")
    end if

end main
```

Bitmaps

1. Introduction
Bitmaps, or in other words, pictures can also be created within an Aptilis programm. You can then save them as GIF files, or output them as GIF pictures to the standard output. This last option is the one needed when you want to create pictures on the fly in web pages.

2. Web mechanics
A lot of HTMLer think that a web page is just one block, composed of text and graphics. The reality is a bit different.
The text of the page, which also contains references to the pictures, is first loaded by the browser. Then, if there are pictures, the browser establishes new connections with the web server in order to retrieve the pictures, one at a time.
That means that the process (for example an Aptilis script) that serves the text of the page, cannot deliver the pictures as well. Although it might be the same piece of code that is responsible of both text and pictures, it's not going to be the same instance of the programm that is going to be involved. In other words, the server is going to run the code several times, once for the text and once for each picture.
Pictures need a special magic line:
Content−Type: image/gif
That means we are going to output pictures in the GIF format. Aptilis cannot output pictures in other formats so far. Have a look at the source of the examples at the bottom of the page, to how this is implemented. It is recommended that you output the header just before the outputGIFBitmap sub, so that if there is any error message, you will see it properly.
Note that if you call a picture from a link, you will not see error messages, but a 'broken image' icon instead. To check what's happening, run your script from the command line. (That's in a DOS box for Windows users: aptilis your_script.e.txt)
The problem seems to be more acute when you're trying a script locally, from your own machine, but in any case, check your script several times.

3. Cache issues
The browser's cache brings problems when playing with pictures. You will probably be confronted to that problem in section a of the next paragraph.
What happens is that when the return of a form is a picture, the browser takes the result from its

cache rather than from the network. The problem seems to solved when calling the script from a link, as in 4.b below, but that's just apparent. The browser indexes its cached element by their URLs. So that
/cgi–bin/aptilis.exe?file=shapes.e.txt /> and
/cgi–bin/aptilis.exe?file=shapes.e.txt /> are completly different altogether if only by a letter.
Unfortunately, if you are working on a script, and you want to try a graphic generating script, even if you call your script from a link, you might not see the changes you've just made. To see them, you will need to reload the frame in which your graphic sits.

4. Examples
Try these example online!

a. Straightforward picture

```
<FORM action="/cgi-bin/aptilis.exe" method="GET">
<INPUT type="hidden" name="file" value="/home/scripts/shapes.e.txt">
Which geometric shape do you want?<BR>
<INPUT type=radio name=corners value=0> Circle<BR>
<INPUT type=radio name=corners value=3> Triangle<BR>
<INPUT type=radio name=corners value=4> Square<BR>
<INPUT type=submit value="Go!">
</FORM>
```

b. calling a picture generating script from links

```
<A href="/cgi-bin/aptilis.exe?file=/home/scripts/shapes.e.txt&corners=4">Square</A><BR>
<A href="/cgi-bin/aptilis.exe?file=/home/scripts/shapes.e.txt&corners=5">Pentagon</A><BR>
<A href="/cgi-bin/aptilis.exe?file=/home/scripts/shapes.e.txt&corners=6">Hexagon</A><BR>
```

c. Generating the page *and* the graphics.

```
<A href="/cgi-bin/aptilis.exe?file=/home/scripts/shape_wrapper.e.txt">List shapes</A>
```

5. The sources of the examples
The shape generator:

```
sub main

        if corners < 3 and corners <> 0

                // For the error message, we will output text.
                print("Content-type: text/plain\n\n")
                print("You specified an incorrect number of corners. Please don't cut corners. Please try again.")

        else

                b = Createbitmap(150, 150)
                if b = -1
                        print("Content-type: text/plain\n\n")
                        print("Sorry, it was not possible to create a bitmap.")
                else
```

```
                        print("Content-type: image/gif\n\n")

                        white = RGB(255, 255, 255)
                        black = RGB(0, 0, 0)
                        clearBitmap(b, white)

                        if corners = 0
                                ellipse(b, 75, 75, 60, 60, black)

                        else
                                twoPi = 3.141592654 * 2
                                ox = 75 + 60
                                oy = 75

                                for i=1 to corners
                                        x = 75 + cos(twoPi * i / corners) * 60
                                        y = 75 - sin(twoPi * i / corners) * 60
                                        line(b, ox, oy, x, y, black)
                                        ox = x
                                        oy = y
                                end for
                        end if
                        outputGIFBitmap(b)

                end if

        end if

end main
```

The wrapper script:

```
sub main

        print("Content-type: text/html\n\n")

        print("<HTML><BODY bgcolor=#FFFFFF>\n")

        for i=0 to 10
                if i=1 or i=2
                        continue
                end if

                // This is the line that will cause the browser to load the pictures
                print("<P><IMG src=\"http://localhost/cgi-bin/aptilis.exe?")
                print("file=f:/aptilis/scripts/shapes.e.txt&corners=", int(i)$, "\"><BR>\n")
                print(int(i)$, " corners\n")
        end for

        print("</BODY></HTML>")

end sub
```

How to make a script runnable

1. Introduction
If you're using aptilis for administrative jobs, you might be tired of typing:
\aptilis\aptilis.exe scriptName.e.txt
Wouldn't it be nice to just type something like:
scriptName ?
It can be done, but it's a slightly differently trick under Windows than it is under Unix.
We will use the following example (hello.e.txt) in both cases:

```
sub main(args[])

        print("That's my programm\n")
        print("Neat, eh?\n")

end main
```

2. Under Windows
In both cases we need to call the interpreter.
Here is how we do it under Windows:
First we need to change the code slightly:

```
@echo off
\aptilis\aptilis.exe hello.bat %1 %2 %3 %4 %5 %6 %7 %8 %9
goto fin

sub main(args[])

        print("That's my programm\n")
        print("Neat, eh?\n")

end main

:fin
```

the 3 first lines disable batch output, call the interpreter and then cause the DOS interpreter to finish execution. What we do is to have an aptilis script being interpreted by the DOS command interpreter, but since this one does not understand aptilis, we have to hide the aptilis code from it, hence the 'goto'.
Then you have to rename your file, in this case 'hello.e.txt' to 'hello.bat'. Files whose extensions are

'.bat' or '.cmd' are automatically passed to the command interpreter, and that's why you can now type 'hello' (without the quotes) at the command line and have your program run straightaway!
The "%1 %2 %3 %4 %5 %6 %7 %8 %9" is necessary to pass possible command line parameters to the script (they can be accessed via *args[]* – see the sub example 5)

This works because aptilis won't run anything outside a sub, so we can put what we want around them.
Unfortunately, unlike with Unix, this trick won't work for web scripts, although the people from Imatix who do the **Xitami** web server assure me that the Windows version of their product understands the **#!** trick as well as the Windows version of the Apache Web Server does.

3. Under Unix
First, the program needs to be modified:

```
#!/HostedWebSites/glaine/cgi-bin/aptilis.exe

sub main(args[])

        print("That's my programm\n")
        print("Neat, eh?\n")

end main
```

The first line instructs your shell what interpreter to use with that file. It's a bit simpler than under Windows, because Unix has been designed to do such things. You don't even need to change the name of your script!
Then, you need to tell Unix that this file can be run from the command line. You do that (for example in a telnet session) like this:
chmod 755 hello.e.txt

This works because aptilis won't run anything outside a sub, so we can put what we want around them.
The good thing about that is that the trick also works for web scripts, and you can call your script directly in the action parameter of your form tag, and no hidden file needs to be passed.

Uploading files from web forms

All browsers worth their salt now support file uploading.
In order to support this in your applications, you need to make sure you do the following:
1. Ensure the required fields are present in your HTML form.
2. Handle the data received in your script.

1. HTML form requirements.
Here's what your form tag should look like:

```
<FORM action="/url/to/aptilis.exe" method="POST" ENCTYPE="multipart/form-data">
<INPUT type="hidden" name="file" value="/path/to/your/aptilis/script.e.txt">
...
</FORM>
```

Note in particular the ENCTYPE attribute set to "multipart/form−data". This tells the browser to send the data in a slightly different way than usual, to cater for possibly bigger, non−text, data chunks.

Now inside this form, you need to offer the user the capability to specify a file (or more) to upload:

```
<INPUT type="file" name="parcel">
```

Did you know this Input type? Well it's required for file uploads. On the HTML page it will look like a text box, with a browse button attached to it − unfortunately for designers, this button cannot be replaced by a graphic.
Example:

2. Handle the data received in your script.
Now, as you would expect from a normal Aptilis script, you will get the entire uploaded file data in the variable 'parcel' − as per our example above. Of course you can name this variable whatever you want.
Now, in addition no normal Aptilis handling of web form fields, 'parcel' is actually an array. (And with Aptilis 'parcel' is the same as 'parcel[0]').
And in 'parcel[1]', you get the filename of the file sent to you. Of course the filename could be important for you to have if the file uploaded is destined to be copied in a repository.

Code example:

```
filename = parcel[1] $
```

Remote sub invocation

1. Introduction
Remote sub invocation allows to call a sub in another aptilis program that runs on the same machine, or on a different machine, on your Intranet or on the Internet.

2. The predefined subs that do it
The subs that allow you to use the remote sub invocation mechanism are:
– TakeCalls()
– Call()
TakeCalls will block the program until it is called by another program.
By default, RSI (Remote Sub Invocation) uses port 1108, but you can change that to any valid port value (1 – 65535) through the second optional parameter of TakeCalls(). However, you need to be superuser on most Unix systems to be allowed to use ports below 1024.

3. Scope
There are plenty of things you can do with that! There is the obvious 'Talk' program listed on the Call() page, but you can also imagine dividing a complex task onto several computers, and then query them through a call to an 'aptilis watchdog' that could tell, for example, if such or such file has been created.
I use it for the word search of my website at http://village.glaine.net/.
I have an aptilis script that has indexed all the words of the site and which then sits idle, waiting for calls.
The script requested by the search page calls the dormant script which returns a list of all the pages where the searched word appears. Because the indexer stays in memory, no disk operation is required for the search so it's quite fast!
For the moment however, calls are replied to sequentially, not asynchronously, in other words, if several calls are sent at the sent time, they will be queued and answered to one after the other.
Depending on how people use this feature, I may implement a multi–threaded version of aptilis that could answer several calls at the same time.

User wrapping (suExec)

1. Introduction
(This is a UNIX only feature).
Depending on how your Web server is setup, you may find that all the files your scripts write (By using the saveFile command for example) are written by someone oddly named 'nobody'. This happens to CGI scripts, and it's not a problem until you want to download or upload those files with **FTP** or you want to access them with **vi**.
It is not only annoying, it is also quite dangerous:
other users of the same sever may read, write or even delete the files your script has written. On top of that you have to allow your target directory to be written to by anyone so that 'nobody' can also save the files in there.
That leaves your space open to anyone else having access to the server to pour tons of data in **your** space...

This happens because the system administrator has set up the web server to run as user 'nobody' to minimize security risks.
(I won't elaborate on the pros and cons of that approach)
In other words, when someone on the web fills a form on your web site, hits [submit] and runs a script, the script is not run by 'you', but by the web server software acting as user 'nobody'.

The solution to this is to enable user wrapping. You can use things such as CGI−wrap if you want to, but aptilis has a built in mechanism to do the same thing. So that when an aptilis script is run, it runs with the same permissions as it's owner's (yours).

2. How to do it
Setting up aptilis to use the wrapping facility is straightforward, but you need to have administrator rights to do it. If you don't, then you'll have to beg him/her to do so. If you are a system administrator make your mind up when you install aptilis the first time: when users have thousands of files written by 'nobody', enabling wrapping will break their scripts, suddenly unable to read the very files they have written in a former life...
To enable wrapping, the aptilis interpreter should have both the sUID and the gUID bits set, and it should belong to user and group 'root'. You do that with the commands:

```
chown root aptilis.exe
chgrp root aptilis.exe
chmod 6755 aptilis.exe

chown root aptilis−run.exe
chgrp root aptilis−run.exe
chmod 6755 aptilis−run.exe (The one that runs aptilis p−code)
as opposed to:
chmod 755 aptilis.exe (to make it a simple executable)
```

Warning: You have to have logged in as root for these to work!

When this has been done, aptilis will run **as** the user owning the aptilis script. Note that in order to run a script as 'Root', both real user ID and effective user ID AND both real group ID and effective group ID have to be root. In other words, unless your web server is itself running as root −and that's a major security risk, a CGI script can never run as root. To run a script as root you need to do it from the command line having logged in as root.

The way aptilis does things is sure easier to set−up and faster than CGI−wrap (CGI−wrap is an extra piece of software called by the web browser. CGI−wrap then calls your script and it's interpreter with the desired permissions). However, since CGI−wrap is a tad more fussy on security, you might still want to use it instead of using the built−in wrapping in aptilis. This is Okay, just remember to **not** set the sUID and the gUID bits on the aptilis file permissions.

Testing your script

If you just want to test your script for errors, but don't want to run it (maybe to prevent it from sending emails etc.), you can use the "compile only flag".

From the command line, adding an exclamation mark at the end of the aptilis file will signify Aptilis to check the program for errors, but not to run it. (You can't have aptilis program file names ending with an '!' now).

Example 1:

```
sub main(args[])

        print( "Hello )

end main
```

save this file as test.e.txt, now type
aptilis.exe test.e.txt!

Result:

```
Content-type: text/plain

print( "Hello )
       ^
At line 3 in sub main Unexpected end of expression, Maybe one or several ')' or
']' or '"' are missing. (code: 26)
```

Example 2:

```
sub main(args[])

        print( "Hello" )

end main
```

aptilis.exe test.e.txt!

Result:

Example 3:

```
sub main(args[])

        print( "Hello" )

end main
```

aptilis.exe test.e.txt *<-- no exclamation mark this time*

Result:

```
Hello
```

Persistence of data across web forms

Warning:
Native session support has been removed with Aptilis 2.4 RC1.
Please use *apt.util.Session* from the library instead.

1. Introduction

Session persistence has to do with CGI applications, or in other words, web based applications.
One annoying thing about web applications, is that your aptilis program (or a program written in any other language) is called repeatedly after each form submission, but it's not the same program (or the same instance of that program) that keeps running from one form to another.
In other words, from one form to the next in −say a booking procedure, all data is lost. To retain it, you must pass it along in hidden fields.
Aptilis remedies that with the '_Session variable'. This variable (which can be an array) keeps its data across sessions.

2. Restrictions

In some cases, although it may appear ideal, it is better not to use the '_Session' variable.

- The '_Session' variable is cleared after an hour, but its time stamp is reset each time a script is run. So a session may last longer than an hour, provided the user doe not stay idle for more than an hour.
- The '_Session' variable relies on an IP number that uniquely identifies the person connecting to your server. In some rare cases, this number can not be retrieved and the '_Session' variable will not be saved.
- Some ISPs (Internet Service Providers) do not give their customers a fixed IP number. This can be a problem when a connection is lost and the customer has to re−dial. Also, several PCs sharing a connection behind a NAT system will share one IP number between them. So from your point of view, your script may confuse data between unrelated users!
- If the person connecting to your server has several browser windows opened, you can be confused as those different windows present you with the same IP number. They might be in different stages of your booking process, or even in a completly different application altogether.
- The persistence mechanism relies on having write access to a temporary directory, or the directory where the aptilis executable resides. Under NT, that will generally not be a problem, but with a Unix server, your scripts may not have write access by default to a temporary directory. This however may be arranged with your system administrator.

In conclusion, a professional, reliable application should rely on hidden fields passed along the different forms of your web application. You can download a example script which uses session ids and cookies from our script archive. However, the '_Session' variable allows you to develop prototypes much faster, or can even be considered seriously if you have control over the network where your application is used.
Control means that you can ascertain that people's IP numbers don't change, and that -should you need to- you can modify read/write permissions on the server you're using.
You usually have this kind of control when you program for an Intranet (a private network using Internet protocols and software) as opposed to the Internet.
Check the Session example.

3. Troubleshooting and technical details

<u>A. Under Windows 9x/NT/2K/XP</u>
When saving session data, aptilis will look for the following environment variables:
1. tmp
2. temp
3. the default directory where aptilis is, at launch time.

You can check the value of the environment variables through the aptilis global array '_ENV'. It's a key based array, so to get the value for the 'tmp' variable, you need to do:

```
sub main

        print("Content-type: text/plain\n\n")
        .
        .
        .
        print("tmp: ", _ENV["tmp"]$, "\n")

end main
```

and you can check the directory you're in with 'getCurrentDirectory'. Aptilis will select the first directory offered by the environment variables or will fall back on its default directory if 'temp' and 'tmp' don't exist. The chosen directory is called the 'Session Directory'.

<u>B. Under Unix</u>
The Unix version of aptilis can only use one directory:
/tmp/aptilis
This directory must be writable by the user the webserver runs as, or by everyone, but that's not recommended, as basically this allows scripts to write tons of data, leaving you with little control over what's happening.
Note that if the /tmp/aptilis/ directory does not exist, or the write permissions are wrong, the session persistance feature of aptilis is disabled with negligible effects on performance.

<u>C. general information</u>
Aptilis keeps its information tidy in a file called 'aptilis-cgi-sessions.dat' in the Session Directory. Several other files with names following the pattern 'sessXXXX.dat' where 'XXXX' is a hexadecimal

serial number are also kept in the Session Directory.
If errors occur, they are kept in an error log file called 'aptilis−cgi−session−errors.txt' which you can check. However, if you do not have write permission to the Session Directory aptilis will not be able to log the errors!
Your troubleshooting should start with finding out where aptilis wants to save it's session data (tmp? temp? the default directory?), then check write permissions and finally analysing the error log.
Note that you can reset all session data by erasing the 'aptilis−cgi−sessions.dat' file.

Examples

- Hello World
- Hello World (from a link)
- Hello World (from a form)
- Passing fields
- More Aptilis
- Sending mail (1)
- Sending mail (2)
- Playing with databases
- Playing with databases − wild cards
- Playing with databases − templates
- Feed−back form
- Doing graphics on the fly
- A real life example
- A web−based newsgroup
- A web page counter
- Persistent data across web forms: Sessions
- Passing data between forms
- A guestbook
- A WAP application
- Server Side Includes
- Using fillForm

Hello World

This example is quite straight forward!
You can see that, as opposed to other languages, it doesn't take three consultants just to print something!!

For this example, you need to have a copy of the interpreter for your machine. The interpreter is 'aptilis.exe'. (Yes, even under Unix, I have left the 'exe' extension.) First type the following text file, for example with the 'Notepad' under Windows or 'vi' under Unix and save it as 'hello1.e.txt'.
You have to be careful with the Notepad which will add a '.txt' extension from time to time. Just make sure you use the right name when you call Aptilis, which is not fussy with the names of the files it is fed with...

```
sub main

        print("Hello World!")

end main
```

Then, run the programme. To do that, you need to pass your programme to the interpreter. You have to make sure that the interpreter is in the path, or that it is in the same directory as your Aptilis programme. Note that Aptilis is a 32 bits programme and will run only under Windows 95 or Windows NT, or, of course under the supported UNIX versions.

DOS/**Windows**, from a DOS box:
aptilis hello1.e.txt

Unix:
aptilis hello1.e.txt *If aptilis is in the path*
./aptilis.exe hello1.e.txt *If aptilis is only in the same directory*

Things learnt:

- The code must be in a 'sub' called main
- You write things with print

Hello World (from a link)

Try online Examples

Now we are into the web!!
People rarely think of calling scripts from a link, rather than from a form. It's a useful thing to keep in mind and here's how to do it! Only caveat is, some servers will not accept a lot of data this way. 200 characters is about the maximum length your URL should be, to be on the safe side.

Here's how we did it:

```
<A href="/cgi-bin/aptilis.exe?file=/home/scripts/hello2.e.txt">Hello?</A>
```

And now, the Aptilis programme, saved as 'hello2.e.txt'

```
sub main

        print("Content-type: text/plain\n\n")
        print("Hello World!")

end main
```

Note that the link should be something like *http://www.a_company.com/cgi−bin/etc...* but I am using what's called a relative link, or a link that's relative to the current URL of the HTML page currently on the browser. I do that because I first try those pages at home, where I have Microsoft's personal Web server installed on my machine and then transfer all the pages to my LINUX server. Using relative links allows me to keep to a minimum of changes when I move the files across to a server with a different URL.

Things learnt

- Doing a script on the web is very simple, but one must **NOT** forget the magic line: print("Content−type: text/plain\n\n")
  This one indicates we will send some simple text, not some HTML. See the next examples...
- The Aptilis programme needn't be in the same directory as the interpreter (aptilis.exe), as long as the directory has got the necessary read permissions (that's under Unix and NT).

Hello World (from a form)

Try online

Examples

This time we will be calling the Aptilis programme from a form.
Here's what the HTML looks like:

```
<FORM action="/cgi-bin/aptilis.exe" method="POST">
<INPUT type="hidden" name="file" value="/home/scripts/hello2.e.txt">

<INPUT type="submit" value="Hello?">
</FORM>
```

And now, the Aptilis programme, saved as 'hello2.e.txt', it's actually the same as the previous one.

```
sub main

        print("Content-type: text/plain\n\n")
        print("Hello World!")

end main
```

Things learnt

- When calling the script form a form, the URL of your Aptilis interpreter must be in the 'action' parameter of your form tag.
- The path to the Aptilis programme must be indicated in a hidden form field, as in:

Passing fields

Try online Examples

Here is how field data is retrieved in an Aptilis programme.

Here's what the HTML looks like:

```
<FORM action="/cgi-bin/aptilis.exe" method="POST">
<INPUT type="hidden" name="file" value="/home/scripts/simple.e.txt">

Say Hello (or something)!:<BR>
<INPUT type="text" name="usersaid" size=20>
<INPUT type="submit" value="Send">
</FORM>
```

And now, the Aptilis programme saved as simple.e.txt

```
sub main


        print("Content-type: text/html\n\n")


        print("<HTML>\n<BODY>\n\n")
        print("<B>you said:</B> ")

        print(usersaid$)

        print("</BODY>\n</HTML>")


end main
```

Things learnt

- Passing form data is straightforward, field data is put in variables for you, so you don't have to do anything!
- To output some real HTML, the magic line needs to be: print("Content-type: text/html\n\n")
- At the end of a sub, instead of 'end sub', you can use the sub's name, after 'end'.

Note:
From version 1.043, aptilis can also retrieve files as sent through the <INPUT type="file" name="varname"> tag. In order to do that, the <FORM> tag of your form must also contain the encoding type used in the following fashion:
ENCTYPE="multipart/form-data"
The method **has** to be 'POST'.

To progress a bit in Aptilis, here is how to calculate your age in seconds. Provided you were born **after** January 1st, 1970...

Here's what the HTML looks like:

```
        <FORM action="!(_AptilisURL)" method="POST">
        <INPUT type="hidden" name="file" value="!(_ScriptsRepository)age.e.txt">

        The day you were born:<BR>
        <INPUT type="text" name="day" size=10><BR>
        <BR>
        The month you were born:<BR>
        <INPUT type="text" name="month" size=10><BR>
        <BR>
        The year you were born:<BR>
        <INPUT type="text" name="year" size=10><BR>
        <BR>

        <INPUT type="submit" value="Get my age in seconds">
        </FORM>
```

And now, the Aptilis program...

```
sub main

        print("Content-type: text/html\n\n")

        // The next line is to clear the array, however
        // all variables are initialized to empty values
        // anyway in Aptilis.

        clearArray(ta[])

        ta[3] = day
        ta[4] = month
        ta[5] = year
        if ta[5] < 100
                // For years entered as 'yy' instead of '19yy'
                ta[5] = ta[5] + 1900
        end if

        print("<HTML>\n<BODY>\n\n")
        print("your age in seconds is: ")

        age = dotime(ta[])
        tn = gettime()

        print(int(tn - age)$)

        print("<BODY>\n</HTML>")

end main
```

See also: doTime, getTime.

Things learnt

- Arrays need not be declared before being used. They grow automatically as new things are added to them.

Sending mail (1)

Try online Examples

You might think that sending an e−mail involves a lot of coding...
Think again!
For this example to work, you need to be connected to the Internet.
***Very importantly** you also need to know an SMTP server that will trust you.*
If you're working from home, you'll find the address of your SMTP server in the settings of your e−mail program.
Most SMTP servers are now more careful about who they accept connections from due to much abuse having been perpetrated by spammers.

See sendMail for more details.

Here's what the HTML looks like:

```
        <FORM action="!(_AptilisURL)" method="POST">
        <INPUT type="hidden" name="file" value="!(_ScriptsRepository)sender1.e.txt">

        Your e-mail address:<BR>
        <INPUT type="text" name="from" size=40><BR>
        <BR>
        Your message:<BR>
        <TEXTAREA name="message" cols=40 rows=5></TEXTAREA><BR>

        <INPUT type="submit" value="Send Message">
        </FORM>
```

And now, the Aptilis programme, sender1.e.txt:

```
sub main

        print("Content-type: text/html\n\n")

        // The value I use here for my SMTP server probably wont work for you.
        // Ask your ISP or your System administrator for an SMTP server to use.

        setSMTPServer("smtp.ntlworld.com")

        print("<HTML>\n<BODY>\n\n")

        sendmail("nirvana@aptilis.com", from$, "Aptilis e-mail example", message$)
        print("Your message has been sent!")

        print("<BODY>\n</HTML>")

end main
```

Things learnt

- There is a simple instruction for sending an e−mail message from an Aptilis programme: sendmail

Sending mail (2)

Try online Examples

Extrapolating on the previous example, we will now see how to retrieve the data from multiple 'select' fields.

The message would be sent to nirvana@aptilis.com

Here's what the HTML looks like:

```
        <FORM ACTION="/cgi-bin/aptilis.exe" METHOD="POST">
        <INPUT TYPE="hidden" NAME="file" VALUE="/home/scripts/sender2.e.txt" />

        Your e-mail address:<BR />
        <INPUT TYPE="text" NAME="from" SIZE="40" /><BR />
        <BR />
        Subject(s):<BR />
        <SELECT MULTIPLE="multiple" NAME="subject" SIZE="4">
        <OPTION>General</OPTION>
        <OPTION>Aptilis</OPTION>
        <OPTION>C/C++ scripting</OPTION>
        <OPTION>Deep Space 9</OPTION>
        </SELECT><BR />
        Use [ctrl] + click to select more than one option!<BR />
        <BR />

        Your message:<BR />
        <TEXTAREA NAME="message" COLS="40" ROWS="5"></TEXTAREA><BR />

        <INPUT TYPE="submit" VALUE="Send Message" />
        </FORM>
```

And now, the Aptilis programme: sender2.e.txt:

```
sub main


        // Of course this value works for me,
        // but you may want to ask your ISP or Administrator
        // what SMTP server to use here.

        setSMTPServer("smtp.ntlworld.com")

        print("Content-type: text/html\n\n")
        print("<HTML>\n<BODY>\n\n")


        n = GetArraySize(subject[])

        if n <= 0
                allsubj = "No subject specified" $
        else

                allsubj = "" $
                for i=0 to n - 1
                        allsubj = allsubj + " " + subject[i] $
```

```
                end for

        end if

        print("The subjects you have chosen are: ", allsubj$, "<br>\n")


        sendmail("nirvana@aptilis.com", from$, allsubj$, message$)
        print("Your message has been sent!")

        print("</BODY>\n</HTML>")

end main
```

Things learnt

- Multiple selections are returned in arrays.

Playing with databases

Here's how to play with databases...
The HTML looks like:

```
<FORM action="/cgi-bin/aptilis.exe" method="POST">
<INPUT type="hidden" name="file" value="/home/scripts/db1.e.txt">

<H3>Products Database Search</H3>

<B>Products</B><BR>
<SELECT name="sel">
<OPTION>Ferrari</OPTION>
<OPTION>PC</PC>
<OPTION>Pizza</OPTION>
</SELECT>

<INPUT type="submit" value="Query...">
</FORM>
```

And now, the Aptilis program...

```
sub main()

        // Magic line.
        print("Content-Type: text/html\n\n")

        // The begining of the page.
        print("<HTML>\n")
        print("<HEAD><BASE href=\"http://localhost/aptilis/\"></HEAD>\n\n")
        print("<BODY bgcolor=#FFFFFF>\n")

        n = LoadDataBase(db[], "f:/aptilis/scripts/database.txt")
        print(int(n)$," record(s)<BR>\n")

        if n < 1
                print("The database is empty</BODY></HTML>")
                return 0
        end if

        // Remember that sel is the name of our SELECT field.
        x = getRecordIndexByKey(db[], sel$, 0)

        if x = -1
                print("<B>Sorry, the product is not in the database.<BR>\n")
        else

                // x is the index of our record.
                r = db[x] $

                print("<IMG src=\"gifs/", GetField(r$,3)$, ".gif\">\n")
                print("<HR>\n")
                print("<B>", getField(r$, 0)$, ":</B> ")
                print("£", getField(r$, 1)$, "<BR>\n")
                print("<I>", getField(r$, 2)$, "</I><BR>\n")
        end if
```

```
        print("</BODY></HTML>\n")

end main
```

Things learnt

- LoadDatabase, to load a database.
- GetRecordIndexByKey, to find a record.
- GetField, to isolate a field in a record.

Here is what our database looks like, it's just a text file, that could have been exported by, say, Access. Or you could have typed it in a text editor.
In this example, I have saved it as "c:\webshare\wwwroot\aptilis\examples\database.txt", that is in the same directory as my web pages. In real life, you might want to store a database in another directory to prevent web users to access it directly from their browser. Under Unix and NT, you must make sure that authorisations are set up correctly.

```
"Ferrari","100000","car","testa"
"PC","1000","computer","pc"
"Pizza"  ,"5.89","food","pizza"
```

Playing with databases – wild cards

Try online Examples

Here, you will see hot to use an approximate search criteria
The HTML looks like:

```
<FORM action="/cgi-bin/aptilis.exe" method="POST">
<INPUT type="hidden" name="file" value="/home/scripts/db2.e.txt">

<B>Products Database Search</B>
<I>By approximation</I><BR>

<INPUT type="text" name="search" size=20>
<INPUT type="submit" value="Search...">

</FORM>
```

And now, the Aptilis programme...

```
sub main()

        print("Content-Type: text/html\n\n")

        print("<HTML>\n")
        print("<HEAD>\n<BASE href=\"http://localhost/aptilis/\"></HEAD>\n\n")
        print("<BODY bgcolor=\"#FFFFFF\">\n")

        n = LoadDataBase(db[], "f:/projects/aptilis/documentation/targets/local/scripts/database.txt")
        print(int(n)$," record(s)<BR>\n")

        if n < 1
                print("The database is empty</BODY></HTML>\n")
        return 0
        end if

        // We are going to use a loop to get ALL the matching records.
        n = 0
        x = 0
        repeat

                x = GetRecordIndexByNearKey(db[], search$, 0, x)
                if x != -1

                        r = db[x] $

                        // Now we output the HTML.
                        print("<IMG src=\"gifs/", getfield(r$, 3)$, ".gif\"><BR>\n")
                        print("Product: ", getfield(r$,0)$, "<BR>\n")
                        print("Prize: &pound;", getfield(r$, 1)$, "<BR>\n")
                        print("<HR>\n")
                        n = n + 1
                        x = x + 1
                end if

        until x = -1

        print("Found: ", int(n)$, " record")
```

```
        if n > 1
                // This adds an 's' at the end of records
                // to be grammatically correct in case of plural.
                print("s")
        end if

        print("<BR>\n\n</BODY></HTML>\n")

end main
```

Things learnt

- GetRecordIndexByNearKey, to find a record in a database, by approximation.
- To do approximate searches, use the special characters '?' for any one letter, and '*' for any group of letter.

"blue"	will match 'blue', exactly.
"f*"	All fields begining with 'f'.
"*i"	All fields ending with 'i'.
"*a*"	All fields containing the letter 'a', but 'apple' doesn't match as 'a' must neither be the first nor the last letter.
"?ree"	All words of any first letter ending in 'ree', like 'tree' or 'free'.
etc...	

As a reminder, here is what our test database contains:

```
"Ferrari","100000","car","testa"
"PC","1000","computer","pc"
"Pizza","5.89","food","pizza"
```

Playing with databases – templates

Try online Examples

Find out how to simplify your life when writing HTML by using 'stuff...

We'll implement a products database search by approximation (e.g.: when the user enters **f*** all products starting with an 'f' will be returned)

Here's what the HTML looks like:

```
<FORM action="/cgi-bin/aptilis.exe" method="POST">
<INPUT type="hidden" name="file" value="/home/scripts/db3.e.txt">

<B>Products Database Search</B> <I>By approximation</I><BR>

<INPUT type="text" name="search" size=20>
<INPUT type="submit" value="Search...">

</FORM>
```

And now, the Aptilis program...

```
sub main()

        print("Content-Type: text/html\n\n")

        print("<HTML>\n")
        print("<HEAD><BASE href=\"http://localhost/aptilis/\"></HEAD>\n\n")
        print("<BODY bgcolor=\"#FFFFFF\">\n")

        n = LoadDataBase(db[], "f:/projects/aptilis/documentation/targets/local/scripts/database.txt")
        print(int(n)$, " record(s)<BR>\n")

        if n = 0
                print("The database is empty</BODY></HTML>")
                return 0
        end If

        // The file that contains the template for a table row.
        t = LoadFile("f:/aptilis/scripts/row.html") $

  print("<TABLE width=50% border=1>\n")

  n = 0
  x = 0
  repeat

    x = GetRecordIndexByNearKey(db[], search$, 0, x)
    if x <> -1

      getAllFields(f[], db[x] $)
      print(stuff(t$)$)

      n = n + 1
      x = x + 1
    end if
```

```
  until x = -1

  print("</TABLE>\n")

  print("Found: ", int(n)$, " record")
  if n>1
    print("s")
  end if

  print(" found.<BR>\n\n</BODY></HTML>\n")

end sub
```

Here is the content of f:/aptilis/scripts/row.html, our template:

```
<TR>

<TD valign=top><IMG src="!(_em)f[3]).gif"></TD>
<TD valign=top bgcolor="#A00000"> <B>!(_em)f[0])</B> </TD>
<TD valign=top>£!(_em)f[1]) </TD>

</TR>
```

As you can see, it is a simple text file which actually contains some HTML. The places where the variables must be inserted are indicated with the syntax: **!(_em)***varname***)**.
See stuff for more details.

Things learnt

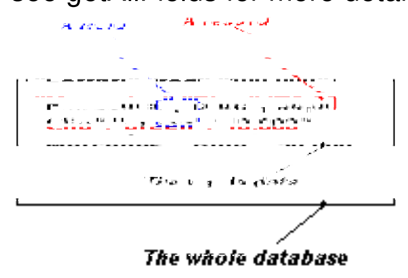
- LoadFile, to load a file directly into a variable.
- stuff, to insert the values of variables into a string.
- When we need to get all the fields of a record into an array, the obvious way to do it would be:

  ```
  for i=0 to 3
    f[i] = GetField(db[x] $, i) $
  end for
  ```

  And that would work, but, there's a faster way to do it:

  ```
  getAllFields(f[], db[x]$)
  ```

  see getAllFields for more details.



As a reminder, here is what our test database contains, it's just a text file, that could have been exported by, say, Access.
In our example, it had been saved as "c:\\aptilis\\database\\database.txt"

```
"Ferrari","100000","car","testa"
"PC","1000","computer","pc"
"Pizza"  ,"5.89","food","pizza"
```

Feed-back form

Try online Examples

This will demonstrate how economically a feedback form can be implemented.
If you're trying that script from home, make sure you are connected to the Internet.
See sendMail for configuration details.

Two things worth noting in this eaxmple. One, how we specify and retrieve a multiple selection from a SELECT HTML object. If you are on a PC you can select more than one option by pressing the [Ctrl] key while clicking options. All the selected options will be available in an array in the Aptilis script. Two, the use of join to concatenate (glue together) the values of the array.

Here's what the HTML looks like:

```
<FORM action="/cgi-bin/aptilis.exe" method="POST">
<INPUT type="hidden" name="file" value="/home/scripts/feedback.e.txt">

Are you a programmer?<BR>
<INPUT type="radio" name="role" value="(Submitted by a programmer)"> Yes<BR>
<INPUT type="radio" name="role" value=""> No<BR>
<BR>
Which topic(s) would like to be more developped in
the examples?<BR>
<SELECT name="topics" size=3 MULTIPLE> (Use [CTRL]+Click to select more than one)
        <OPTION>Real life examples</OPTION>
        <OPTION>Database usage</OPTION>
        <OPTION>Math applications</OPTION>
        <OPTION>Web based forms</OPTION>
        <OPTION>Graphics</OPTION>
</SELECT>
<BR>
<BR>
Notes:<BR>
<TEXTAREA name="notes" ROWS=5 COLS=40></TEXTAREA>
<BR>
<BR>
<INPUT type="text" name="email" size=40><BR>
<BR>
<INPUT type="submit">
</FORM>
```

And now, the Aptilis programme...

```
sub main

        print("Content-Type: text/html\n\n")

        message = role + "\n\n" $

        n = getarraysize(topics[])
        if n > 0
                message = message + "Topics the sender would like to see more developped:\n" $
        end if

        // Note the use of the join predefined sub that
```

```
        // joins together the members of an array.
        message = message + join(topics[], "\n") + "\n" $

        if len(notes$)
                notes = replace(notes$, "\r", "") $
                message = message + "\nNotes:\n-----\n" + notes $
        end if

        print("<HTML><BODY>Your message reads:\n\n<PRE>", message$, "</PRE>\n")
        print("<B>Thank you ", email$,"</B>.\n")

        print("</BODY></HTML>")

        // Remember, this works for me here, but ask your ISP
        // or your system administrator for a valid SMTP server to use.
        setSMTPServer("smtp.ntlworld.com")
        sendmail("teebo@glaine.net", email$, "Aptilis feed-back form", message$)

end main
```

Things learnt

- How to retrieve the data from different kind of fields.

Doing graphics on the fly

This example will demonstrate the kind of things you can do with the ability Aptilis has to generate pictures. We will calculate biorythms.
Biorythms are based on the assumption that humans go through cycles. (I am not necessarily endorsing that, but it's a nice programming example.) There's an emotional cycle that lasts 28 days, a physical one that goes on for 23 days, and an intellectual one that spans 33 days. In short that means that every 23 days you are at the top of your physical shape. The cycles start the day you're born.

Here's what the HTML looks like:

```
<FORM action="/cgi-bin/aptilis.exe" method="GET" target="biograph">
<INPUT type=hidden name=file value="/home/scripts/biorytms.e.txt">

<B>Day of birth (1, 2, ...30, 31)</B> <INPUT type="text" size="8" name="day"><BR>
<B>Month of birth (1...12)</B> <INPUT type="text" size="8" name="month"><BR>
<B>Year of birth (ex: 1970)</B> <INPUT type="text" size="8" name="year"><BR>
<BR>
<INPUT type=submit value="Check biorythms">
</FORM>
```

And here's the Aptilis source:
This programme caused me a few headaches as either my local server, some hidden proxy on the web or even the server would defeat it by caching the generated pictures. So trying a date would work, but any subsequent test would still bring the same original bio−rythm curve. I solved the problem by using the GET method. It puts the data from the form in the URL, and that can be limited in size, but at least that makes for a different URL for each different date, and no more caching problem. Since we are not sending a lot of data, GET is fine here.

```
sub main

        _twoPi = 2 * 3.141592654

        if year > 1900
                year = year - 1900
        end if

        // The time functions won't go before 1970, so we implemented our own
        nDays = getDays(day, month, year) - 7

        b = createBitmap(400, 250)
        if b = -1
                print("Content-type: text/plain\n\n")
                print("Sorry, it was not possible to create a bitmap...")
        else
                print("Content-type: image/gif\n\n")
                white = RGB(255, 255, 255)
                black = RGB(0, 0, 0)

                red = RGB(255, 0, 0)
                green = RGB(0, 255, 0)
                blue = RGB(0, 0, 255)
```

```
		cleargray = 11
		gray = 10
		setColor(b, gray, 160, 160, 160)
		setColor(b, cleargray, 192, 192, 192)


		clearBitmap(b, white)
		box(b, 0, 0, 399, 249, gray, 0)
		box(b, 2, 10, 397, 194, cleargray, 1)

		line(b, 67, 4, 73, 4, gray)
		line(b, 70, 4, 70, 200, gray)
		box(b, 48, 200, 92, 220, gray, 0)
		printAt(b, 50, 217, "today", blue)
		line(b, 70, 220, 70, 246, gray)
		line(b, 67, 246, 73, 246, gray)

		for i=1 to 39
			x = i * 10
			line(b, x, 241, x, 246, gray)
		end for

		for i=1 to 4
			printAt(b, 77 + i * 70, 217, "+" + int(i * 7)$, gray)
		end for

		dl = int(nDays + 7) + " days lived" $
		getStringMetrics(b, dlm[], dl$)
		birth = format("0", 2, 0, day) + "/" + format("0", 2, 0, month) + "/" + int(year + 1900) $


		DoCurve(b, nDays, red, 28)
		DoCurve(b, nDays, blue, 23)
		DoCurve(b, nDays, green, 33)

		printAt(b, 399 - dlm[0] - 2, 26, dl $, black)
		printAt(b, 316, 42, birth$, black)


		printAt(b, 4, 26, "Cycles:", black)
		printAt(b, 12, 48, "Emotional", red)
		printAt(b, 12, 68, "Physical", blue)
		printAt(b, 12, 88, "Intellectual", green)

		outputGIFBitmap(b)
		deleteBitmap(b)

	end if

end main


sub DoCurve(b, n, colour, p)


	// to avoid calculations on large values, which breaks the sin function on some platforms.
	n = (n % p) - 1

	ox = 2
```

```
        oy = 102
        x = 2
        for i=.2 to 39.7 step .1
                y = 102 - sin( (i + n) * _twoPi / p) * 92
                line(b, ox, oy, x, y, colour)
                ox = x
                oy = y
                x = x + 1
        end for

end DoCurve


sub getDays(d1, m1, y1)

        // The time functions won't go before 1970, so we'll implement our own.

        m[1] = 0
        m[2] = 31
        m[3] = 59
        m[4] = 90
        m[5] = 120
        m[6] = 151

        m[7] = 181
        m[8] = 212
        m[9] = 243
        m[10] = 274
        m[11] = 304
        m[12] = 334

        t = getTime()
        FillTimeArray(ta[], t)

        q1 = d1 + m[m1] + y1 * 365.25
        q2 = ta[3] + m[ta[4] + 1] + (ta[5] - 1900) * 365.25

        if (y1 & 3) = 0 and m1 > 2
                q1 = q1 + 1
        end if

        if (ta[4] & 3) = 0 and ta[4] > 2
                q2 = q2 + 1
        end if

        return q2 - q1

end getDays
```

A real life example

How to replace a string by another in a series of files...

I had started to do those examples on my laptop and then moved them to my usual workstation. The two computers have different IP addresses and that affected the <FORM action="..."> tag in some of my examples.
After considering different options, (Manual replacement,C, PERL...) I found that Aptilis was actually the most indicated solution.
(No, really that's how it happened!)

```
sub main

        // That's the IP of my laptop
        origin = "10.0.0.44" $

        // and the one my desktop uses
        new = "192.168.2.4" $

        // Just get all the files we want
        n = getFileList(fl[], "*.html")

        for i=0 to n-1

                // We have a file name in fl[i]
                print(fl[i]$,"\n")

                // Now, we just load the file into f$
                f = loadfile(fl[i]$) $

                // If you wanted to, you could keep a backup.
                // savefile(fl[i] + ".bak" $, f$)

                // Here we replace the bit we want to replace
                f = replace(f$, origin$, new$) $

                // ...and we put the file back on the disk.
                savefile(fl[i]$, f$)

        end for

end main
```

A web-based newsgroup

Try online Examples

This shows that Aptilis is well at ease when it comes to fairly complex tasks. Here is how to implement a web-based discussion forum...
Even the Aptilis Forums are based on the original code!

Much credit goes to Steven de Brouwer for having fixed many quirks in that script.

```
sub Init()

        // Here we initialize all the values we will need all over the application.
        // remember that variable names starting with an underscore are global to the
        // whole program. Ie. once defined, they can be seen in all the subs.


        _scriptURL = "http://localhost/cgi-bin/aptilis.exe" $
        _scriptPath = "f:/projects/aptilis/documentation/targets/local/scripts/newsgroup.e.txt" $
        _articlesPath = "c:/temp/aptilis/" $

        if right(_articlesPath$, 1) = "/" $
                _articlesPath = left(_articlesPath$, len(_articlesPath$) - 1) $
        end if

        // To notify me of any posting.
        setSmtpServer("smtp.ntlworld.com")


end Init



sub AddButton(command, face, addition)

        // This sub adds a button, in effect a complete form.

        print("<FORM action=\"", _scriptURL$, "\" method=\"POST\" target=\"article\">\n")
        print("<INPUT type=\"hidden\" name=\"file\" value=\"", _scriptPath$, "\">\n")
        print("<INPUT type=\"hidden\" name=\"cmd\" value=\"", command$, "\">\n")

        print(addition$)

        print("<CENTER><INPUT type=\"submit\" value=\"", face$, "\"></CENTER>\n")
        print("</FORM>\n")

end AddButton



sub NewPostingForm(a, m)


        print("<BODY bgColor=\"#FFFFFF\">\n")


        if len(a$) > 0
                // This is a reply, we need to load the article.
```

```
		ChangeDirectory(_articlesPath$)
		file = loadFile(a$)$

		n = separate(lines[], file$, "\n")
		file = "" $

		print("<TABLE border=1><TR><TD valign=top><B> Reply")
		if m = 1
			print(" and Mail")
		end if
		print(" To:</B></TD><TD>")
		print(lines[2]$, "</TD></TR>\n")

		if len(trim(lines[1]$)$) = 0
			name = "[Anonymous]" $
		else
			name = lines[1]$
		end if


		print("<TR><TD>by:</TD><TD>")
		if len(lines[0]$)
			print("<I><A href=\"mailto:", lines[0]$, "\">")
		end if

		print(name$)

		if len(lines[0]$)
			print("</A>")
		end if

		print("</I></TD></TR></TABLE>\n<HR>\n\n")

		content = "" $
		for i = 3 to n - 1
			content = content + "> " + replace(lines[i]$, "\r", "") + "\n" $
		end for

		title = replace("Re: " + lines[2]$, "\"", "'") $

	else
		print("<B>Post new article</B><HR><P>")
	end if



	print("<FORM action=\"", _scriptURL$, "\" method=\"POST\">\n")
	print("<INPUT type=\"hidden\" name=\"file\" value=\"", _scriptPath$, "\">\n")
	print("<INPUT type=\"hidden\" name=\"cmd\" value=\"30\">\n")
	print("<INPUT type=\"hidden\" name=\"a\" value=\"", a$, "\">\n")
	print("<INPUT type=\"hidden\" name=\"m\" value=\"", m$, "\">\n")

	print("<TABLE>")

	print("<TR><TD><B>Title of Posting:</B></TD>\n")
	print("<TD><INPUT type=\"text\" name=\"title\" Value=\"", title$, "\" size=20></TD></TR>\n")

	print("<TR><TD valign=top><B>Posting:</B></TD>\n")
	print("<TD><TEXTAREA name=\"posting\" cols=50 ROWS=10>", content$, "</TEXTAREA></TD></TR>\n")
```

```
        print("<TR><TD valign=top><B>Your e-mail:</B></TD>\n")
        print("<TD><INPUT type=\"text\" name=\"email\" size=50></TD></TR>\n")

        print("<TR><TD valign=top><B>Your name:</B></TD>\n")
        print("<TD><INPUT type=\"text\" name=\"name\" size=50></TD></TR>\n")


        print("<TR><TD></TD><TD><INPUT type=\"submit\" value=\"Post article\"></TD></TR>\n")
        print("</TABLE>\n\n")

        print("</FORM>\n\n")
        print("</BODY>\n")


end NewPostingForm



sub AddPosting(title, posting, email, name, a, m)

// <I>the 'posting' and 'a' variables come directly from the form.</I>


        // <I>To remove those **** carriage returns from PCs...</I>
        posting = replace(posting$, "\r", "") $


        // <I>To be sure that those are only one line</I>
        // <I>Otherwise we might not put the information where it should be...</I>
        email = replace(email$, "\r" ,"") $
        email = replace(email$, "\n", " ") $
        name = replace(name$, "\r", "") $
        name = replace(name$, "\n", " ") $
        title = replace(title$, "\r", "") $
        title = replace(title$, "\n", " ") $


        // <I>The file we are going to create will have this format:</I>
        // <I>line 0: email</I>
        // <I>line 1: name</I>
        // <I>line 2: title</I>
        // <I>line 3 and after: the posting</I>


        if len(name$) = 0
                name = email $
        end if


        total = email + "\n" + name + "\n" + title + "\n" + posting $


        changeDirectory(_articlesPath$)

        if len(a$) = 0


                // Find a file name: this is a new posting.
                t = gettime()
```

```
		fillLocalTimeArray(date[], t)
		d = int(date[6]) $
		d = string(3 - len(d$), "0") + d $

		serial = 0
		repeat

			serial = int(serial) $
			s = string(3 - len(serial$), "0") + serial $

			filename = path + d + s + "-.txt" $
			serial = serial + 1

		until fileExist(filename$) = -1

	else
		// This is a reply to an existing posting.


		// We need to load the file in order to send a reply to the orginal poster it if requested.
		if m = 1
			file = loadFile(a$)$
			n = separate(lines[], file$, "\n")
			file = "" $
			if len(lines[0]$) > 0
				sendmail(lines[0]$, "teebo@glaine.net", "Aptilis Test Neswgroup - Reply to " + title$, posting$)
			end if
		end if


		p = instr(a$, "-") $
		d = left(a$, p - 1) $

		serial = 0
		repeat

			serial = int(serial) $
			s = string(3 - len(serial$), "0") + serial$
			filename = path + d + s + "-.txt" $
			serial = serial + 1

		until fileExist(filename$) = -1

	end if



	// ...And finally, save the posting.
	savefile(filename$, total$)

end AddPosting



sub DisplayFiles


	if ChangeDirectory(_articlesPath$) = -1
		print("Impossible to reach the Postings' repository: ", _articlesPath$, "<BR>")
		print("Errno: ", _errno$, "<BR>")
```

```
		return
	end if


	// Because of the way the names of the articles
	// are created, sorting them in alphabetical order
	// puts them in the right order for the display.

	n = GetFileList(fl[], "*.txt")
	sortarray(fl[], "alphabetical")

	// Display the entries


	print("<UL>\n")
	l = 2
	for i=0 to n - 1

		nl = (instr(fl[i]$, "-") -1) / 3
		bcl = nl

		while bcl < l
			print("</UL>\n")
			bcl = bcl + 1
		end while

		while bcl > l
			print("<UL>\n")
			bcl = bcl-1
		end while

		DisplayEntry(fl[i]$)
		l = nl

	end for
	print("</UL>\n")


end DisplayFiles



sub DisplayEntry(f)

	file = loadFile(f$) $
	n = separate(l[], file$, "\n")

	// <I>We don't need this anymore, let's save memory</I>
	file = "" $


	// <I>We want to be sure to have all the fields!</I>
	if n < 3
		return 0
	end if


	print("<LI><B><A href=\"", _scriptURL$, "?")
	print("file=", _scriptPath$, "&cmd=40&a=", f$, "\" target=\"article\">")
	print("[", l[2]$, "]</A>")
```

```
	for i=0 to 2
		lens[i] = Len(l[i]$)
	end for



	// <I>Put the Author's name, if any</I>
	if lens[1]
		print(" - ", l[1]$)
	end if

	print("</B> (", int( GetFileSize(f$) - lens[0] - lens[1] - lens[2] - 2)$, " chars)")
	print("</LI>\n")

end DisplayEntry



sub DisplayPosting(a)

	// This sub displays a single Posting in it's entirety.

	print("<BODY bgColor=\"#FFFFFF\">\n\n")

	changeDirectory(_articlesPath$)
	file = loadFile(a$) $

	n = separate(lines[], file$, "\n") $
	file = "" $


	print("<B>", lines[2]$, "</B><BR>\n")

	if len(lines[0]$)
		print("By <I><A href=\"mailto:", lines[0]$, "\">")
		print(lines[1]$, "</A></I>\n<HR>\n\n")
	end if


	print("<PRE>")
	for i = 3 to n - 1
		// The original line feeds have been removed by LoadDatabase but there might be some CRs
		s = replace(lines[i]$,">","&gt;") $
		s = replace(s$,"<","&lt;") $
		print(s$, "\n")
	end for

	print("</PRE>\n\n")


	// The buttons
	print("<CENTER><TABLE><TR><TD bgcolor=#00C000>")

	add = "<INPUT type=\"hidden\" name=\"m\" value=0>\n<INPUT type=\"hidden\" name=\"a\" value=\"" + a + "\">\n" $
	AddButton("20", "Reply Only", add$)

	print("</TD><TD bgcolor=#0000C0>")
```

```
        add = "<INPUT type=\"hidden\" name=\"m\" value=1>\n<INPUT type=\"hidden\" name=\"a\" value=\"" + a + "\">\n" $
        AddButton("20", "Reply + Mail", add$)

        print("</TD></TR></TABLE></CENTER>\n\n")
        print("</BODY>\n")


end DisplayPosting



sub main

        // To initialize all our necessary values.
        Init()

        // Magic line!
        print("Content-Type: text/html\n\n")



        // Common to everything we will be doing.
        print("<HTML>\n<HEAD>\n\t<TITLE>Aptilis Demo News-Group</TITLE>\n\t<BASE href=\"", _Base$, "\">\n</HEAD>\n\n")



        // We use a variable, cmd, to decide what to do.
        select cmd
        case 0
                // Display the Frameset.
                print("<FRAMESET COLS=\"30%,*\">\n")
                print("    <FRAME SRC=\"", _scriptURL$, "?file=", _scriptPath$, "&cmd=10\" NAME=\"plist\">\n")
                print("    <FRAME SRC=\"", _scriptURL$, "?file=", _scriptPath$, "&cmd=5\" NAME=\"article\">\n")
                print("</FRAMESET>\n\n")
                break

        case 5
                // Entry screen.
                print("<BODY bgColor=\"#FFFFFF\">\n")
                print("<TABLE width=100% height=100%><TR><TD vAlign=center align=center><FONT Face=\"Arial,Helvetica\"><IMG Src=\"http://www.aptilis.com/aptilis-tiny.gif\" Alt=\"Aptilis\"><BR>Demo Newsgroup</FONT></TD></TR></TABLE>\n\n")
                print("</BODY>\n")
                break

        case 10
                // List all Postings in left hand frame.
                print("<BODY bgColor=\"#FFFFFF\">\n")

                print("<HR><CENTER><IMG Src=\"http://www.aptilis.com/aptilis-tiny.gif\" Alt=\"Aptilis\"></CENTER><HR>\n")
                AddButton("20", "Post Something", "")
                DisplayFiles()

                print("</BODY>\n")
                break

        case 20
                // Article Form.
                NewPostingForm(a$, m$)
                break

        case 30
```

```
		// Add Posting.
		AddPosting(title$, posting$, email$, name$, a$, m$)
		print("<BODY>Thanks for your posting.\n\n")

		// We use a hidden form and some javaScript to refresh the list of Postings.
		print("<FORM Action=\"", _scriptURL$, "\" Method=\"Post\" Target=\"plist\" Name=\"ngrfh\">\n")
		print("<INPUT Type=\"hidden\" Name=\"file\" Value=\"", _scriptPath$, "\">\n")
		print("<INPUT Type=\"hidden\" Name=\"cmd\" Value=\"10\">\n")
		print("</FORM>\n\n")

		print("<SCRIPT Language=\"javaScript\">\n<!--\ndocument.ngrfh.submit()\n//-->\n</SCRIPT></BODY>")
		break

	case 40
		// Display a single posting.
		DisplayPosting(a$)
		break

	end select



	print("</HTML>\n")

end main
```

A web page counter

Try online

Examples

Counters are very popular CGI applications, and are also a useful tool to easily keep an eye on how well a page is doing.
The counter above has been implemented in Aptilis. Several things make Aptilis a great tool to develop counters.
They are:

- Simplicity, no need to know how to create a GIF at the binary level!
- Built−in graphic support.
- True Type support, ie. nice fonts.
- Locking mechanism.

Note that you don't have to actually display the hit count, you can use any graphic or a white pixel. This allows you to keep an idea of the number of hits for a particular page without telling the public.

Hit refresh to see the number of hits increasing. Nothing might happen if you are accessing the net through a proxy that will insist on caching the graphic. Unfortunately, that's not something Aptilis − or any other language, can address.

Here is how the picture above is called:

```
<IMG src="/cgi-bin/aptilis.exe?file=/home/scripts/counter.e.txt">
```

That's how I did it on this machine. Of course */home/scripts/counter.e.txt* is the script below, and you should replace that and */cgi−bin/aptilis.exe* by appropriate values, ie. *http://www.yourserver.com/cgi−bin/aptilis.exe*.

Here's the source:

```
sub main

        fontPath = "c:/winnt/fonts/times.ttf" $
        path = "c:/temp/aptilis/eq-counter.dat" $

        // Let's try to get a lock on the hits file.
        maxWait = 10
        t = getTime()
        while lock(path$) = -1
                if getTime() - t > maxWait
                        failed = 1
                        break
                end if
        end while

        if failed = 1
                hits = "oops" $
        else
                // load the hits, and increment them
                hits = loadFile(path$)
                saveFile(path$, int(hits +1)$)
```

```
                // give it back
                unlock(path$)
        end if

        // Now we can take our time with the graphic
        b = CreateBitmap(114, 20)
        if b != -1

                print("Content-type: image/gif\n\n")

                // White bacground
                clearBitmap(b, 255)

                // anything is possible, but OK, I won't get any design awards on that one!
                if p = 0 or p = 5

                        if p = 0
                                box(b, 0, 0, 15, 10, RGB(255, 0, 0), 1)
                                box(b, 0, 0, 15, 10, 0, 0)
                        end if

                        // now that's the point: let's use a nice font!
                        setFont(b, fontPath$, 14)

                        mess = int(hits) + "hits" $
                        getStringMetrics(b, mf[], mess$)
                        printAt(b, (114 - mf[0]) / 2, 15, int(hits) + " hits"$, 0)
                end if
                outputGIFBitmap(b)

                // because we're tidy
                deleteBitmap(b)

        else
                print("Content-type: text/plain\n\n")
                print("It was impossible to create the bitmap.\n")
        end if

end sub
```

Things learnt

- Fonts.
- Locking mechanism, lock doesn't block.

Persistent data across web forms: Sessions

Try online Examples

Warning:
Native session support has been removed with Aptilis 2.4 RC1.
Please use *apt.util.Session* from the library instead.

This example illustrates how simply you can go with forms by using the _Session global variable which is kept across forms. Before you embark upon using that feature like crazy, make sure you know the whole story by reading 'Persistence of data across web forms' as there are a few restrictions.

The example given here is of a typical process to get a sales prospect's details.
The example has taken it to the extreme and you will note that the 'Back' button of your browser defeats the process. The cure to that, is of course, to pass the stage the user's in as a hidden field rather than storing it in _Session[0].

Here's the source of the aptilis program:
Note how quite simple this is!

```
sub main


        print("Content-Type: text/html\n\n")
        print("<HTML>\n<HEAD>\n\t<TITLE>Session based form</TITLE>\n</HEAD>\n\n")

        print("<BODY bgcolor=\"#FFFFFF\">\n\n")


        if reset
                _Session[0] = 0
        end if



        print("<FORM action=\"http://localhost/cgi-bin/aptilis.exe\" method=\"POST\">\n")
        print("<INPUT type=\"hidden\" name=\"file\" value=\"f:/projects/aptilis/documentation/targets/local/scripts/session.e.txt\">\n\n")


        select _Session[0]
        case 0

                _Session[0] = 1
                print("<B>1. Personal details</B><BR>\n")
                print("Please input your name: <INPUT type=\"text\" name=\"s1Name\"><BR>\n")
                print("<INPUT type=\"submit\" value=\"next\"><BR><BR><BR>\n")

                break

        case 1
                _Session[0] = 2
                _Session[1] = s1Name$


                print("<B>2. Quick contact</B><BR>\n")
```

```
                print("Please input your e-mail: <INPUT type=\"text\" name=\"s2Email\"><BR>\n")
                print("<INPUT type=\"submit\" value=\"next\"><BR><BR><BR>\n")
                break

        case 2
                _Session[0] = 3
                _Session[2] = s2Email$
                print("<B>3. Where to send the goods</B><BR>\n")
                print("Please input your postal address: <TEXTAREA name=\"s3Addr\" cols=30 rows=6></TEXTAREA><BR>\n")
                print("<INPUT type=\"submit\" value=\"Finish\"><BR><BR><BR>\n")
                break

        case 3
                _Session[0] = 0
                print("The following information:\n")
                print("<TABLE border=1><TR><TD>")
                print("Name: ", _Session[1]$, "<BR>\nE-mail: ", _Session[2]$, "<BR>\nAddress: ", s3Addr$)
                print("</TD></TR></TABLE><BR>has been sent to the Jelly&copy; Corp. Of San Blobino, Ca. for assessment. You should receive your jelly sample in the post before the end of this decade or the next.")

        end select

        print("</FORM><BR>\n<BR>\n")



        print("</BODY></HTML>")

end main
```

Things learnt

- Persistent data is kept in the _Session array, a global variable (As the '_' indicates)
- This approach has drawbacks and limitations, see The 'Session' topic

Passing data between forms

Try online

Examples

Here we're going to see how to pass data across several forms.
Here are the different techniques we're using in that example:

- We use one script, instead of 5, and we use a variable, *c*, to indicate at what stage we are.
- We pass the data previously collected across the different stages in hidden form fields.
  Example:

  ```
  <INPUT type="hidden" name="firstname" value="Teebo">
  ```

- We print the whole forms directly, instead of using templates and the stuff command, which would be more flexible on a design point of view.
- We call a script from a link:

  ```
  <A href="/cgi-bin/aptilis.exe?file=/home/scripts/multi_forms.e.txt&c=1">Try...</A>
  ```

Here's the source of the aptilis program:

```
sub Init
        // Using an Init sub is a good idea to centralize
        // Initializations, especially if your script is going
        // to move between several servers.


        _ScriptUrl = "http://localhost/cgi-bin/aptilis.exe" $
        _ScriptPath = "f:/projects/aptilis/documentation/targets/local/scripts/multi_forms.e.txt" $

end Init


sub main

        Init()
        print("Content-type: text/html\n\n")


        // In our example all the forms have this part in common
        // Note the use of the c variable, though

        print("<HTML>\n")
        print("<HEAD>\n\t<TITLE>Our form, stage ", int(c)$, "</TITLE>\n</HEAD>\n")
        print("<BODY bgcolor=\"#FFFFD0\">\n")


        print("<H1>Our form, stage ", int(c)$, "</H1>\n")

        print("<FORM action=\"", _ScriptUrl$, "\" method=\"POST\">\n\n")
        print("<INPUT type=\"hidden\" name=\"file\" value=\"", _ScriptPath$, "\">\n")
```

```
// We use a control variable to know what to do
select c
case 1
        // The next stage
        print("<INPUT type=\"hidden\" name=\"c\" value=\"2\">\n\n")

        // We get their first name
        print("Please enter your first name:<BR>\n")
        print("<INPUT type=\"text\" name=\"firstname\" size=30><BR>\n")

        break

case 2

        // The next stage
        print("<INPUT type=\"hidden\" name=\"c\" value=\"3\">\n\n")

        // This is the BIG SECRET!!
        // How we keep the data across forms
        // We have 'firstname' from the previous form.

        // Double quotes may break the html
        firstname = replace(firstname$, "\"", "'") $
        print("<INPUT type=\"hidden\" name=\"firstname\" value=\"", firstname$, "\">\n\n")


        // We get their first name
        print("Please enter your last name:<BR>\n")
        print("<INPUT type=\"text\" name=\"lastname\" size=30><BR>\n")

        break


case 3

        // The next stage
        print("<INPUT type=\"hidden\" name=\"c\" value=\"4\">\n\n")

        // Keeping the data from previous forms
        lastname = replace(lastname$, "\"", "'") $
        print("<INPUT type=\"hidden\" name=\"firstname\" value=\"", firstname$, "\">\n")
        print("<INPUT type=\"hidden\" name=\"lastname\" value=\"", lastname$, "\">\n\n")

        print("Please type in your address:<BR>\n")
        print("<TEXTAREA name=\"address\" cols=30 rows=5></TEXTAREA>\n")

        break

case 4
        // The next stage
        print("<INPUT type=\"hidden\" name=\"c\" value=\"5\">\n\n")

        // Keeping the data from previous forms
        // Here, we must make sure that carriage returns we may have in the address
        // won't break our HTML

        // get rid of those annoying char 13 coming from PCs
        address = replace(address$, "\r", "") $
```

```
                // and replace new lines by a code we invented, unlikely to appear in an address
                address = replace(address$, "\n", "&nl;") $

                print("<INPUT type=\"hidden\" name=\"firstname\" value=\"", firstname$, "\">\n")
                print("<INPUT type=\"hidden\" name=\"lastname\" value=\"", lastname$, "\">\n")
                print("<INPUT type=\"hidden\" name=\"address\" value=\"", address$, "\">\n\n")


                // the next (and last) part of our form
                print("You are a:<BR>\n")
                print("<INPUT type=\"radio\" value=\"Male\" name=\"sex\" checked> Man<BR>\n")
                print("<INPUT type=\"radio\" value=\"Female\" name=\"sex\"> Woman<BR>\n")

                break

        case 5
                // And finally...
                print("Here are the details you entered:<BR>\n")

                if sex = "Male" $
                        print("Mr ")
                else
                        print("Mrs ")
                end if


                print("<FONT color=\"#D00000\">", firstname$, "</FONT> ")
                print("<FONT color=\"#00D000\">", lastname$, "</FONT><BR>\n")

                // We're transforming address into a displayable piece of HTML text
                address = replace(address$, "<", "&lt;") $
                address = replace(address$, ">", "&lt;") $
                address = replace(address$, "&nl;", "<BR>") $

                print("<FONT color=\"#0000D0\">", address$, "</FONT><BR>\n")


                break

        end select



        // part 5 is not a form. so we don't need a submit button
        if c != 5
                print("<P><INPUT type=\"submit\">\n\n")
        end if


        // but since we're always printing the form header, we need that
        // in order to have a clean HTML page
        print("</FORM>\n")



        print("</BODY>\n</HTML>\n")

end main
```

Things learnt

- Using a variable to indicate what part of the script to run.
- Putting an Init sub at the begining of the program for easy porting.

A guestbook

Try online Examples

A long awaited example, this guestbook is very simple and most of the code has to do with displaying the HTML page.
Note there's no 'break' statement in the case 5 block, and that's on purpose. This illustrates why you need break statements, ie, to be able to omit them if required.

```
sub Init

        _DPath = "c:/temp/aptilis/" $
        _Base = "http://localhost/aptilis/" $
        _ScriptURL = "http://localhost/cgi-bin/aptilis.exe" $
        _ScriptPath = "f:/aptilis/scripts/guestbook_example.e.txt" $
        _admin = "teebo@glaine.net" $

        setSMTPServer("smtp.ntlworld.com")

end Init


sub purify(m)

        m = replace(m$, "\r", "") $
        m = replace(m$, "<", "&lt;") $
        m = replace(m$, ">", "&gt;") $
        m = replace(m$, "\n", "<BR>") $
        m = replace(m$, "\"", "'") $
        return m$

end purify


sub main

        Init()
        print("Content-type: text/html\n\n")

        print("<HTML>\n<HEAD>\n")
        print("\t<TITLE>Aptilis guestbook</TITLE>\n")
        print("\t<BASE href=\"", _Base$, "\">\n")
        print("</HEAD>\n\n")

        print("<BODY bgcolor=\"#FFFFFF\">\n\n")

        print("<FONT face=\"arial,helvetica\" color=\"#0000D0\">")

        print("<FONT size=+2><CENTER>\n\n")


        // Change title here
        print("This is the Aptilis guestbook!<BR><I>Feel free to add your comments!</I><BR>\n")

        print("<FONT size=-1>(Scroll down to the bottom to post a comment)</FONT><BR>\n")
        print("</CENTER></FONT><HR>\n\n")
```

```
select c
case 5
        // Save post

        name = purify(name$) $
        email = purify(email$) $
        comment = purify(comment$) $

        t = int(getTime()) $
        rec = makeRecord(t$, email$, name$, comment$) $
        appendRecord(_DPath + "guestbook-data.dat" $, rec$)

        if len(_admin$) > 0
                sendmail(_admin$, _admin$, "Aptilis Test Guestbook", comment + "\n\n" + name$, email$)
        end if


case 2
        // And Display the contents


        n = loadDataBase(db[], _DPath + "guestbook-data.dat" $)

        // Simple way to print out the last message in first
        reverseArray(db[])

        for i=0 to n - 1
                getAllFields(fls[], db[i]$)
                if len(fls[1]$) > 0
                        eml1 = "<A href=\"mailto:" + fls[1] + "\">" $
                        eml2 = "</A>" $
                else
                        eml1 = "" $
                        eml2 = "" $
                end if

                fillLocalTimeArray(dt[], fls[0])
                date = int(dt[3]) + "/" + int(dt[4] + 1) + "/" + int(dt[5]) $


                print("\"", fls[3]$, "\"<BR><BR>")
                print(eml1$, "<B>", fls[2]$, "</B>", eml2$, " (", date$, ")")
                print("\n\n<HR>\n")

        end for


        print("<FORM action=\"", _ScriptURL$, "\" method=\"POST\">\n")


        print("<INPUT type=\"hidden\" name=\"file\" value=\"", _ScriptPath$, "\">\n")
        print("<INPUT type=\"hidden\" name=\"c\" value=\"5\">\n")

        print("<TABLE>\n")
        print("<TR><TD valign=top><FONT face=\"arial,helvetica\">Your name:</FONT></TD>")
        print("<TD valign=top><INPUT type=\"text\" name=\"name\" size=40></TD></TR>\n")

        print("<TR><TD valign=top><FONT face=\"arial,helvetica\">Your e-mail address:</FONT></TD>")
        print("<TD valign=top><INPUT type=\"text\" name=\"email\" size=40><BR></TD></TR>\n")
```

```
                print("<TR><TD valign=top><FONT face=\"arial,helvetica\">Your comments:</FONT></TD>")
                print("<TD><TEXTAREA name=\"comment\" rows=5 cols=40 wrap></TEXTAREA></TD></TR>\n")

                print("<TR><TD></TD><TD valign=top><INPUT type=\"submit\" value=\"send\"></TD></TR>\n")

                print("</TABLE>\n")
                print("</FORM>")

                break
        end select

        print("</FONT></BODY></HTML>\n")

end main
```

Here's the HTML that would call the guestbook directly. Note that you do not necessarily need a target.

```
<A href="/cgi-bin/aptilis.exe?file=/home/scripts/guestbook.e.txt&c=2" target=_blank>Try this guestbook.</A>
```

A WAP application

Remember the age in seconds example? Well, here's the same as a WAP application!
WML, the language understood by WAP phones is a bit fussier than HTML, so make sure you put everything where it should be, expecially the headers.
A good place to start for WAP is:
http://www.wapforum.org/

Here is how to calculate your age in seconds on your wap phone. Provided you were born **after** January 1st, 1970...

Here's what the **WML** looks like:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"  "http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
<card id="form1">
<p>Day: <input name="day" type="text"/>
Month: <input name="month" type="text"/>
Year: <input name="year" type="text"/>
</p>

<do type="accept" name="Done">

        <go href="/cgi-bin/aptilis.exe" method="get">
                <postfield name="day" value="$day"/>
                <postfield name="month" value="$month"/>
                <postfield name="year" value="$year"/>
                <postfield name="file" value="/home/scripts/age_wml.e.txt"/>
        </go>

</do>

</card>
</wml>
```

And now, the Aptilis programme...

```
sub main

        print("Content-type: text/vnd.wap.wml\r\n\r\n")

        // The next line is just to be tidy.
        clearArray(ta[])

        ta[3] = day
        ta[4] = month
        ta[5] = year
        if ta[5] < 100
                // For years entered as 'yy' instead of '19yy'
                ta[5] = ta[5] + 1900
        end if
```

```
        // This is important - don't omit it
        print("<?xml version=\"1.0\"?>\n")
        print("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\" \"http://www.wapforum.org/DTD/wml_1.1.xml\">\n")


        print("<wml>\n<card id=\"answer\">\n\n")
        print("<p>your age in seconds is: ")


        age = dotime(ta[])
        tn = gettime()


        print(int(tn - age)$)
        print(" - http://www.aptilis.com/")

        print("</p>\n</card>\n</wml>\n")


end main
```

See also: dotime, gettime.

Things learnt

- WML Basics.
- Aptilis does WAP − Tell your friends.

Server Side Includes

Try online Examples

There is another, quite powerful way to generate web pages: Server Side Includes.
Basically you have a normal web page, such as this one, but parts of it (or in this case one part) are generated by one or several scripts.
A SSI page could look like this:

```
<html>
<head>
        <title>Example: Server Side Include</title>
</head>

Here, for example, we have a counter:<BR>
<!--#include virtual="/cgi-bin/ssi.e.txt"-->

<p>
(....)
</body>
</html>
```

And the ssi.e.txt script is as follows:

```
sub main

        // This should lock the file as in the web counter example,
        // but we're keeping things simple here.
        // For locking, check the Web counter example.
        // The advantage of using SSI is that here we're outputing text
        // and not a graphic. In some cases, it may well be more
        // flexible.

        print("Content-type: text/html\n\n")
        n = loadFile("c:/temp/aptilis/ssihits.dat")
        saveFile("c:/temp/aptilis/ssihits.dat", n + 1)
        print("This is another way to generate a hit counter: <FONT color=\"#FF0000\"><B>", int(n)$, "</B></FONT>")

end main
```

Now, in order to get this to work, you need to ensure the following:
– Your file (such as this page) must usually have a different extension than '.html'. On this server, it's '.shtml'.
– You must put your script in the CGI directory and make it runnable.

Unix + This means running the command chmod 755 name_of_your_script.e.txt
+ You must indicate where the interpreter for your file is.
This means that you must specify aptilis.exe as an interpreter for yur script.
To do that, you use the #! syntax on the first line of your script, example:
#!/home/httpserver/cgi–bin/aptilis.exe
See how to make a script runnable in more details.

Windows + Microsoft IIS can do SSI as well, see the IIS help for details
+ If you are using Apache, the configuration is the same as for Unix systems, except that you don't have to chmod the script and the first line of the script should look something

like
#!c:\apache\www\cgi–bin\aptilis.exe
+ When using OmniHTTPd make sure that *Process Server Side Includes (SSI)* is enabled (you'll find it under *Configuration –> Web Server Global Settings –> Advanced*). Then change to the tab *External* and add : *Virtual: .apt* (or e.txt) / *Actual: c:\aptilis\bin\aptilis.exe* (if your aptilis.exe resides in this location)

This is because whatever you're invoking is supposed to come from the same server as the page.
– Your server must be configured to support SSI. I found this to be a bit tricky myself. The syntax used here 'include virtual' may also be different on your server. On my Apache server I had to add this to the access conf.file:

```
<DIRECTORY /home/*/html>
Options +Includes
AllowOverride None
order allow,deny
allow from all
</DIRECTORY>
```

That would enable SSI in the html directories of my users.

Using fillForm

Try online Examples

FillForm is a very powerful Aptilis predefined sub that will save you a lot of time! It is like Stuff only far more powerful as it understands HTML! This time I'll show you the Aptilis code first:

```
sub main

        _FormPath = "f:/aptilis/scripts/htmlform-example.html" $

        // As always...
        print("Content-type: text/html\n\n")

        // are the compulsory fields filled with something?
        // (We could use trim before this in case the user has
        // entered only tabs or spaces)

        if len(forename$) = 0 or len(surname$) = 0 or len(email$) = 0

                // If we come back from the form all the values are here in local variables.
                // eg, name, surname, email, language, gender.

                // the template
                form = loadFile(_FormPath$) $

                // we fill it
                readyForm = fillForm(form$) $

                print(readyForm$) // done!

        else

                // All compulsory fields are filled...
                print("<HTML><BODY>")
                print("Thanks. Your details are now going to be processed.<BR>")
                print("(application goes on from here...)")
                print("</BODY></HTML>")

        end if

end sub
```

Here's what the HTML looks like, it's just a web page:

```
<HTML>

<BODY bgColor="#FFFFD0">

<IMG Src="http://www.glaine.net/images/aptilis-tiny.gif">

<P><B>Demo form!</B><BR>
<I>Don't worry it doesn't keep your details!</I>

<P>


<FORM action="/cgi-bin/aptilis.exe" Method="POST">
```

```
<!-- the usual. It will be passed as well! And put back in! -->
<INPUT type="hidden" name="file" value="c:\aptilis\fillform-example.e.txt">

<B>Fields in bold are compulsory.</B> For as long as they're not filled,
you will becoming back here!
<P><TABLE>
<TR>
        <TD><B>Name:</B></TD>
        <TD><INPUT type="text" size="20" Name="forename"></TD>
</TR>

<TR>
        <TD><B>Surname:</B></TD>
        <TD><INPUT type="text" size="20" Name="surname"></TD>
</TR>

<TR>
        <TD><B>Your e-mail address:</B></TD>
        <TD><INPUT type="text" size="20" Name="email"></TD>
</TR>


<TR>
        <TD>Language:</TD>
        <TD><SELECT name="language">
            <OPTION>English</OPTION>
            <OPTION>French</OPTION>
            <OPTION>Italian</OPTION>
            <OPTION>German</OPTION>
            <OPTION>Spanish</OPTION>
            <OPTION>Korean</OPTION>
            </SELECT></TD>
</TR>

<TR>
        <TD>Gender:</TD>
        <TD><INPUT type="radio" name="sex" value="female"> Woman<BR>
            <INPUT type="radio" name="sex" value="male"> Man</TD>
</TR>

<TR>
        <TD></TD>
        <TD><INPUT type="submit"></TD>
</TR>

</TABLE>

</BODY>
</HTML>
```

Things learnt

- Using fillForm simplifies your Aptilis programs: The HTML code is no longer intertwined with your Aptilis program
- Your Aptilis code will be far more legible
- You can let a designer who does not know programing maintain or change the looks of your application, **provided the designers stick to the same HTML form field names!**

- Just make sure your HTML is clean, FillForm is not a HTML doctor. If something looks weird, it means you have forgotten a " or a **>**, etc... :-)

Predefined Functions

- Advanced
- Arrays
- Bitmaps
- Databases
- Files
- Flow control
- Html
- Input and Output
- Internet
- Loops
- Math Functions
- Miscellaneous
- Streams
- Strings
- Time
- Variables
- XML

Advanced

Security

Security

- Checkmark
- Mark

Checkmark

Security

Aptilis 1

Checkmark(String)

CheckMark allows you to check that a password against a marked directory. You can mark a directory with mark

Mark marks the current directory with a password of your choice.
To mark another directory, use changeDirectory.
The password you choose is encrypted and written in file called 'passw', so you must be careful not to erase this file.
The main use of this fairly low level and weak feature is to allow you to ask someone a password before you might allow them to read or write in a given directory, especially from a web script.
This scheme is independant from the usual UNIX security mechanisms. Someone with the necessary login/password could still write in a marked directory. If security is paramount for your application, you should envisage other ways to do it and possibly discard Aptilis.

Return value:
0 if the passwords match, –1 otherwise.
Note that if there is no passw file (ie. the directory has not been marked) in the directory you're checking, the return will also be –1.

Example:

```
// We are marking the current directory
mark("motDePasse")

rem this one will work
v1 = checkmark("motDePasse")

rem but this one won't...
v2 = checkmark("I don't know the password")

print(v1, "\n", v2)
```

Result:

```
0.0000
-1.0000
```

Mark

Security

Aptilis 1

Mark(String)

Mark marks the current directory with a password of your choice.
To mark another directory, use changeDirectory.
The password you choose is encrypted and written in file called 'passw', so you must be careful not to erase this file.
Use checkMark to check a password.
The main use of this fairly low level and weak feature is to allow you to ask someone a password before you might allow them to read or write in a given directory, especially from a web script.
This scheme is independant from the usual UNIX security mechanisms. Someone with the necessary login/password could still write in a marked directory. If security is paramount for your application, you should envisage other ways to do it and possibly discard Aptilis.

Return value:
0 if the directory has been marked correctly, −1 otherwise.
The most likely error will happen under UNIX, if do not have write permissions in the directory you're trying to mark, check _errno to be sure. In case of problems, you must refer to your sys−admin.

Example:

```
// We are marking the current directory
mark("motDePasse")

// this one will work
v1 = checkmark("motDePasse")

// but this one won't...
v2 = checkmark("I don't know the password")

print(v1, "\n", v2)
```

Result:

```
0.0000
-1.0000
```

Arrays

ClearArray

Arrays

Aptilis 1

ClearArray(ArrayName[][, predifined size])

ClearArray deletes an array. Its size is reset to zero and none of its element is available any more. The optional predefined size parameter can be used to warn Aptilis in advance if you know how many elements the array is likely to contain. That does speed things up.

Return value:
0

Example:

```
a[0] = 22
a[1] = 35


print(a[0], "\n", a[1], "\n")
cleararray(a[])
print("----\n", a[0], "\n", a[1], "\n")
```

Result:

```
22.0000
35.0000
----
0.0000
0.0000
```

See also SetArrayIndex, SetArrayDimensions, GetNext, GetPrevious, GetArraysize.

GetArrayDimensions

Arrays

Aptilis 2

GetArrayDimensions(dest[], Array[])

GetArrayDimensions will retrieve a description of an array.
It is particularly useful with multi-dimensional arrays that have ben declared with SetArrayDimensions.

Return Value:
A list of integers in dest[] describing the size of each dimension of Array[].

Example:

```
setArrayDimensions(squares[], 6, 4)

getArrayDimensions(dims[], squares[])
n = getArraySize(dims[])

print("Dimensions: ")
for i = 0 to n - 1
print(int(dims[i])$, " ")
end for
```

Result:

```
Dimensions: 6 4
```

See also SetArrayDimensions, ClearArray.

GetArraySize

Arrays

Aptilis 1

GetArraySize(ArrayName[])

GetArraySize simply returns the number of elements of an array.
Arrays are created automatically when you first use them. They are made big enough so that they will be able to contain all your data. In the same fashion, you do not need to worry about resizing your arrays, because if you refer to an element outside the boundaries of the array, the array is automatically expanded.
For instance:
a[11] = 5 *An array with 12 elements is created, (Arrays start at 0)*
print(a[2]) *An empty string is returned*
print(a[20]) *Nothing happens, we're just reading, no need to expand*
a[22]= "hello"$ *The array is automatically expanded to 23 elements, all is cool.*

Return value:
array size.

Example:

```
a[0]="Apple" $
a[10]="Banana" $
a[20]="Cherry" $

n = GetArraySize(a[])
print(n)
```

Result:

```
21
```

See also Setarrayindex, GetNext, GetPrevious, ClearArray.

GetNext

Aptilis 1

GetNext(ArrayName[])

GetNext gets the next value of an array. Each array has a hidden index which you can set with SetArrayIndex to access all its values one by one.
Once you've gotten the last element available, the hidden index goes back to the begining of the array.
This is particulary useful with arrays that use string key indexing (rather than numbers), when you do not know the different keys.

Return value:
Next array element.

Example 1:

```
a[0] = "Apple" $
a[1] = "Banana" $
a[2] = "Cherry" $

SetArrayIndex(a[], "begin")
n = getarraysize(a[])
for i = 1 to n
print(getnext(a[])$, "\n")
end for
```

Result:

```
Apple
Banana
Cherry
```

Example 2, a Franco−English fruit dictionary:

```
a["Pomme"] = "Apple" $
a["Banane"] = "Banana" $
a["Cerise"] = "Cherry" $

SetArrayIndex(a[], "begin")
n = getarraysize(a)
for i=0 to n - 1
print(getNext(a[])$, "\n")
end for
```

Result:

```
Apple
Banana
Cherry
```

See also SetArrayIndex, GetPrevious, GetArraysize, ClearArray.

GetNextKey

Aptilis 1

GetNextKey(ArrayName[])

GetNextKey gets the next key of an array. Each array has a hidden index which you can set with SetArrayIndex to access all its values or keys one by one.
Once you've gotten the last element available, the hidden index goes back to the begining of the array.
This is particulary useful with arrays that use string key indexing (rather than numbers), when you do not know the different keys.

Return value:
Next array key.

Example 1:

```
a["one"] = "un" $
a["two"] = "deux" $
a["three"] = "trois" $

SetArrayIndex(a[], "begin")
n = getarraysize(a[])
for i = 1 to n
key = GetNextKey(a[]) $
print("The French for '", key$, "' is '", a[key$]$, "'\n")
end for
```

Result:

```
The French for 'one' is 'un'
The French for 'two' is 'deux'
The French for 'three' is 'trois'
```

See also SetArrayIndex, GetPreviouskey, GetArraysize, ClearArray.

GetPrevious

Arrays

Aptilis 1

GetPrevious(ArrayName[])

GetPrevious gets the previous value of an array. Each array has a hidden index which you can set with SetArrayIndex to access all its values one by one.
Once you've gotten the last element available, the hidden index goes back to the end of the array.
This is particulary useful with arrays that use string key indexing (rather than numbers), when you do not know the different keys.

Return value:
Previous array element.

Example 1:

```
a[0] = "Apple" $
a[1] = "Banana" $
a[2] = "Cherry" $

SetArrayIndex(a[], "end")
n = getarraysize(a[])
for i = 1 to n
print(getprevious(a[])$, "\n")
end for
```

Result:

```
Cherry
Banana
Apple
```

Example 2, a Franco−English fruit dictionary:

```
a["Pomme"] = "Apple" $
a["Banane"] = "Banana" $
a["Cerise"] = "Cherry" $

SetArrayIndex(a[], "end")
n = getarraysize(a)
for i=0 to n - 1
print(getPrevious(a[])$, "\n")
end for
```

Result:

```
Cherry
Banana
Apple
```

See also SetArrayIndex, GetNext, GetArraysize, ClearArray.

GetPreviousKey

Aptilis 1

GetPreviousKey(ArrayName[])

GetPreviousKey gets the previous key of an array. Each array has a hidden index which you can set with SetArrayIndex to access all its values or keys one by one.
Once you've gotten the last element available, the hidden index goes back to the end of the array.
This is particulary useful with arrays that use string key indexing (rather than numbers), when you do not know the different keys.

Return value:
Next array key.

Example 1:

```
a["one"] = "un" $
a["two"] = "deux" $
a["three"] = "trois" $

SetArrayIndex(a[], "begin")
n = getarraysize(a[])
for i = 1 to n
key = GetPreviousKey(a[]) $
print("The French for '", key$, "' is '", a[key$]$, "'\n")
end for
```

Result:

```
The French for 'three' is 'trois'
The French for 'two' is 'deux'
The French for 'one' is 'un'
```

See also SetArrayIndex, GetNextkey, GetArraysize, ClearArray.

ReverseArray

Aptilis 1

ReverseArray(ArrayName[])

ReverseArray reverses the order of the elements in an array.

Return value:
The number of elements in the array.

Example:

```
sub main

a[0] = "dog" $
a[1] = "cat" $
a[2] = "fish" $
a[3] = "snail" $
a[4] = "mouse" $

reverseArray(a[])
n = getArraySize(a[])
for i=0 to n - 1
print(a[i]$, "\n")
end for

end sub
```

Result:

```
mouse
snail
fish
cat
dog
```

See also SetArrayIndex, GetNext, GetPrevious, GetArraysize.

SetArrayDimensions

Arrays

Aptilis 2

SetArrayDimensions(Array[], dim1size, dim2size, ...)

SetArrayDimensions allows you to declare multidimensional arrays. Multidimensional arrays need to be declared and space in memory needs to be reserved for their elements. Aptilis needs to know the size of each dimension in order to calculate where to store a value.
A consequence of this is that you may get errors at run−time, that is in the middle of a program being run. This happens very rarely with Aptilis which is not fussy and tolerates your errors, however an array index out of range will stop a program dead in its tracks.

Return Value:
The total number of elements in the array or 0 if not enough memory was available for the array.

Example:

```
setArrayDimensions(squares[], 6, 4)

for y = 0 to 3
for x = 0 to 5
squares[x,y] = x * y
end for
end for

for y = 0 to 3
for x = 0 to 5
print(format(" ", 3, 0, squares[x,y])$)
end for
print("\n")
end for
```

Result:

```
  0  0  0  0  0  0
0  1  2  3  4  5
0  2  4  6  8 10
0  3  6  9 12 15
```

See also GetArrayDimensions, ClearArray.

SetArrayIndex

Arrays

Aptilis 1

SetArrayIndex(ArrayName[], Index)

SetArrayIndex is used to set the internal index of an array. In aptilis, all the variables are arrays and they have an internal counter which allows retrieval of array values one by one, which is especially useful in stringed key arrays, where the keys are not known in advance.
If you don't set the array index before retrieving values with either GetNext or GetPrevious, there is no way you can foresee which value you are going to get.
You can feed string values to SetArrayIndex, ie. "begin", or "end", or an actual number.
The index given to SetArrayIndex might not correspond exactly to the actual index or reference of the value. Indeed, if you have an array with three values, the indexes you can give to *setarrayindex* might not correspond to the actual indexes:

```
a["one"] = "un" $                    (index -> 0)
a["ten"] = "dix" $                   (index -> 1)
a["one hundred"] = "cent" $          (index -> 2)
```

Example 1:

```
a["pomme"] = "Apple" $
a["banane"] = "Banana" $
a["cerise"] = "Cherry" $

setArrayIndex(a[], "begin")
t = getNext(a[])$
print(t$)
```

Result:

```
Apple
```

Example 2:

```
a["pomme"] = "Apple" $
a["banane"] = "Banana" $
a["cerise"] = "Cherry" $

setArrayIndex(a[], "end")
t = getPrevious(a[])$
print(t$)
```

Result:

```
Cherry
```

Example 3:

```
a["pomme"] = "Apple" $
a["banane"] = "Banana" $
a["cerise"] = "Cherry" $
```

```
setArrayindex(a[], "begin")
for i = 1 to getArraySize(a[])
t = getNext(a[])$
print(t$, "\n")
end for
```

Result:

```
Apple
Banana
Cherry
```

Example 4:

```
a["pomme"] = "Apple" $
a["banane"] = "Banana" $
a["cerise"] = "Cherry" $

setArrayindex(a[], "end")
for i = 1 to getArraySize(a[])
t = getPrevious(a[])$
print(t$, "\n")
end for
```

Result:

```
Cherry
Banana
Apple
```

See also GetNext, GetPrevious, GetArraySize, ClearArray.

SortArray

Arrays

Aptilis 1

SortArray(ArrayName[], stringHow)

SortArray will sort an array. You choose to sort alphabetically or numerically.
You have to keep in mind that all the empty values will be put at the begining of the array.
For key based arrays, the values are also reordered, and that affects how values are retrieved by getNext and getPrevious, and the same keys are still attached to the same values.
The second paremeter is either "numeric" or "alphabetical" to indicate how you want the variables to be sorted.
Use SortDatabase to sort databases.

Return value:
0 if everything is OK.
–1 if there has been a problem and the variable _errno contains the cause of the error.

Example 1:
Sorting an array numerically. Note that strings containing only letters have a value of 0.

```
m[0] = 52.253
m[1] = "1" $
m[2] = "12.5" $
m[3] = 34
m[4] = "Hello" $
m[5] = 1000
m[6] = -27
m[7] = -35
m[8] = "5" $


sortarray(m[], "numeric")
n = getarraysize(m[])

for i=0 to n - 1

print("SORTED: ", m[i]$, "\n")

end for
```

Result:

```
SORTED: -35.000000
SORTED: -27.000000
SORTED: Hello
SORTED: 1
SORTED: 5
SORTED: 12.5
SORTED: 34.000000
SORTED: 52.253000
SORTED: 1000.000000
```

Example 2:
Sorting an array alphabetically.

```
m[0] = "Bonjour" $
m[1] = "Hello" $
m[2] = "Ciao" $
m[3] = "Arivederci" $
m[4] = "Salut" $
m[5] = "Morning!" $
m[6] = "Hasta luego" $
m[7] = "Guten Tag" $
m[8] = "Sayonara" $


sortarray(m[], "alphabetical")

n = getarraysize(m[])
for i=0 to n - 1
print("SORTED: ", m[i]$, "\n")
end for
```

Result:

```
SORTED: Arivederci
SORTED: Bonjour
SORTED: Ciao
SORTED: Guten Tag
SORTED: Hasta luego
SORTED: Hello
SORTED: Morning!
SORTED: Salut
SORTED: Sayonara
```

Example 3:
Sorting a key based array alphabetically.

```
m["Hello"] = "Bonjour" $
m["Bye"] = "Au revoir" $
m["Speak to you soon"] = "A bientôt" $
m["Ta"] = "Ciao" $

sortarray(m[], "alphabetical")

setArrayIndex(m[], "begin")
n = getarraysize(m[])

for i=0 to n - 1
print("SORTED: ", getNext(m[])$, "\n")
end for
```

Result:

```
SORTED: A bientôt
SORTED: Au revoir
SORTED: Bonjour
SORTED: Ciao
```

See also SetArrayIndex, GetPrevious, GetArraySize.

Bitmaps

Box

Aptilis 1

Box(BitmapHandle, x1, y1, x2, y2, Colour, Fill)

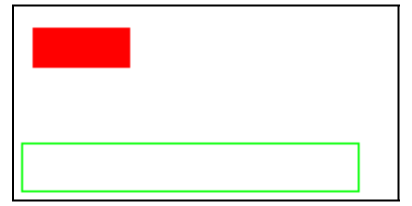
Box allows you to draw a box from (including) (x1, y1) to (including)(x2, y2).
The colour is specified through its index.
The 'Fill' parameter indicates if you only want the frame of the box (ie. a rectangle) or if you want it to be filled as well. Any non−zero value for 'Fill' will make full boxes.

Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap.

Example:

```
b = createBitmap(200, 100)
if b = -1
print("Ooops, could not create the bitmap....\n")
else

// White background
clearBitmap(b, RGB(255, 255, 255))

// The filled red box
box(b, 10, 10, 60, 30, RGB(255, 0, 0), 1)

// The green box
box(b, 4, 70, 180, 95, RGB(0, 255, 0), 0)

saveGIFFile("box.gif", b)

end if
```

Result:



For more details see the Bitmap topic.

ClearBitmap

Aptilis 1

ClearBitmap(BitmapHandle, Colour)

ClearBitmap fills an entire bitmap with one single colour.
The colour is indicated by its index which ranges form 0 to 255.
This instruction is really useful because, when initialized, bitmaps may contain random colours.
ClearBitmap is much faster and a lot less memory−hungry than Fill applied on the whole bitmap.
ClearBitmap also uses the color you have given it to set the background colour for the bitmap. See setBackground.

Return Value:
0 if everything was OK, or −1 if you specified an incorrect bitmap.

For more details, like the Aptilis colour system see the Bitmap topic.

CreateBitmap

Bitmaps

Aptilis 1

CreateBitmap(width, height)

CreateBitmap creates a virtual (ie. in memory) bitmap for you to draw onto.
There are several graphic functions that allow you to set pixels, draw lines, circles, etc...
You can then either save the graphic into a GIF file or output it into the standard output if you want to send a GIF file to a web page through a CGI−script.
A default palette is assigned to the bitmap. Aptilis bitmap can only be of the 256 colour type. You can change the colours if you wish to, but that may ruin the True Type fonts anti−aliasing. See also setColor, RGB, hexColor.

Return Value:
The handle of the new bitmap, or −1 if it wasn't possible to create the bitmap because of a lack of memory or because one or the two dimensions was equal to 0. Check _errno for more details.

For more details see the Bitmap topic.

DeleteBitmap

Aptilis 1

DeleteBitmap(BitmapHandle)

DeleteBitmap removes a previously created bitmap from memory. Bitmaps can be voracious in terms of RAM resources, and freeing some RAM will help other programmes and your own go faster.
A bitmap eats 1 byte per pixel (that's 64Kb for a 320 * 200 bitmap) plus 768 bytes for the palette.
Freeing bitmaps is always a good idea when you're done with them!
Remaining bitmpas are freed automatically when your Aptilis program finishes.

Return Value:
0, or −1 if the handle was incorrect.

For more details see the Bitmap topic.

Ellipse

Aptilis 1

Ellipse(BitmapHandle, xCenter, yCenter, aRadius, bRadius, Colour)

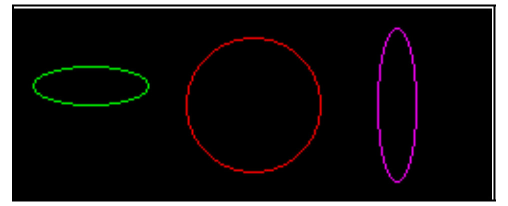
Ellipse will draw ellipses(ovals) and circles.
You need to specify a center, (xCenter, yCenter) and two radiuses. aRadius is the horizontal radius and bRadius is the vertical one.
The trick to draw **circles** is to use identical values for aRadius and bRadius.

Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap.

Example:

```
b = createBitmap(250, 100)
if b = -1
print("Ooops, could not create the bitmap....\n")
else

// black background
clearBitmap(b, RGB(0, 0, 0))


// The green oval
ellipse(b, 40, 40, 30, 10, RGB(0, 255, 0))

// The red circle
ellipse(b, 125, 50, 40, 40, RGB(255, 0, 0))

// The Magenta oval
ellipse(b, 200, 50, 10, 40, RGB(255, 0, 255))


saveGIFFile("ellipse.gif", b)

end if
```

Result:



For more details see the Bitmap topic.

Fill

Aptilis 1

Fill(BitmapHandle, x, y, Colour, StopColour)

Fill is the command to use to fill a region with a particular colour.
The 'stopColour' parameter indicates the border of the zone to fill. But Fill won't go over a line drawn in 'colour' either. Fill tends to be slower than ClearBitmap, and requires much more memory, so to fill an entire bitmap, ClearBitmap is a better choice.

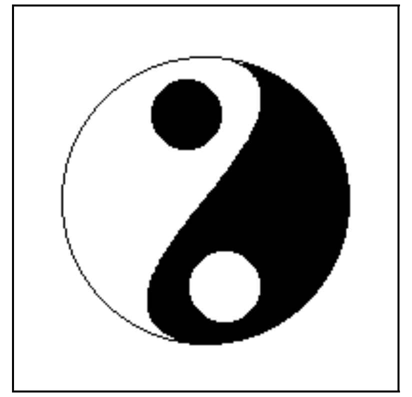
Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap.

Example:

```
b = createBitmap(200, 200)
if b = -1
print("Ooops, could not create the bitmap....\n")
else

// White background
clearBitmap(b, RGB(255, 255, 255))


// The body
ax = 30
pi = 3.141593654
st = pi * 2 / 150

black = RGB(0, 0, 0)
white = RGB(255, 255, 255)

ellipse(b, 100, 100, 75, 75, black)

oy = 25
ox = 100
a = 0

for i=0 to 149
nx = 100 + sin(a) * ax

ny = 100 - cos(a / 2) * 75

line(b, ox, oy, nx, ny, black)

ox = nx
oy = ny
a = a + st
end for


// The small black circle
ellipse(b, 90, 55, 18, 18, black)
fill(b, 90, 55, black, black)
```

```
// The small white circle
// We could do a black circle first, but I want to illustrate the stop colour.
fill(b, 100, 120, black, black)
ellipse(b, 110, 145, 18, 18, white)
fill(b, 110, 145, white, white)



saveGIFFile("yy.gif", b)


end if
```

Result:



For more details see the Bitmaps topic.

GetPixel

Aptilis 1

GetPixel(BitmapHandle, x, y)

GetPixel allows you to check the colour of a pixel in a previously created bitmap.

Return Value:
The colour of the pixel or −1 if you indicated a non−valid bitmap.

For more details see the Bitmap topic.

GetStringMetrics

Aptilis 1

GetStringMetrics(BitmapHandle, DestinationArray, String)

GetStringMetrics returns the size in pixels of rectangle large enough to contain the specified string. The result is returned in an array, where the first element is the width of the string and the second, its height.

Return Value:
0 and a width and a height in the specified array or −1 if you indicated a non−valid bitmap.

Example:

```
b = createBitmap(250, 50)
if b = -1
print("Ooops, could not create the bitmap....\n")
else

// White background
clearBitmap(b, RGB(255, 255, 255))

// Unix user need to find a true type font and copy its file
// somewhere on their system, and refer to it here correctly.
// Windows user can choose another ttf file if they want.

setFont(b, "c:\\windows\\system\\aribl0.ttf", 20)

getStringMetrics(b, mf[], "Aptilis is ")
getStringMetrics(b, ms[], "COOL!")

// total width
w = mf[0] + ms[0]


x = (250 - w) / 2
// The y coordinate is the bottom of the string, hence the plus.
y = (50 + mf[1]) / 2


printAt(b, x, y, "Aptilis is", RGB(0, 0, 0))
printAt(b, x + mf[0], y, "COOL!", RGB(0, 255, 255))

saveGIFFile("font.gif", b)


end if
```

Result:

For more details see the Bitmap topic.

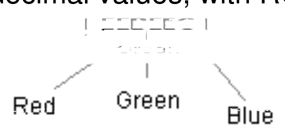
HexColor

Bitmaps

Aptilis 1

HexColor(HexadecimalString)

HexColor transforms a string that represents a colour, into an Aptilis colour index. Aptilis uses a default 256 colour palette, which contains a broad range of tones. HexColor takes a string such as "FFDE5C" which contains three hexadecimal values and translates the values for red, green and blue into an Aptilis color index that matches the given colour in the best possible way.
Of course if you have redefined some or all of the 256 available colours, HexColor will most probably return an unsuitable value.
Hexadecimal colour values are very often used in web pages.
Unless you have an hexadecimal value ready for use, you may prefer to use more meaningful decimal values, with RGB
.



Return Value:
An Aptilis colour index. If any part of the string contains an invalid character, then the remaining colours are considered to be 0.
Note: Valid hexadecimal numbers are numbers from 0 to 9 and letters from A to F. Two hexadecimal numbers make up a value which ranges from 0 to 255.

Example:

```
b = createBitmap(90, 200)
if b = -1
print("Oops...\n")
else

clearBitmap(b, RGB(255, 255, 255))


printAt(b, 4, 20, "Red", RGB(255, 0, 0))
printAt(b, 4, 40, "Green", RGB(0, 255, 0))
printAt(b, 4, 60, "Blue", RGB(0, 0, 255))
printAt(b, 4, 80, "Yellow", RGB(255, 255, 0))



printAt(b, 4, 100, "HxRed", hexColor("FF0000"))
printAt(b, 4, 120, "HxGreen", hexColor("00ff00"))
printAt(b, 4, 140, "HxBlue", hexColor("0000FF"))
printAt(b, 4, 160, "HxYellow", hexColor("FFFF00"))
printAt(b, 4, 180, "HxMagenta", hexColor("FF00FF"))
printAt(b, 4, 199, "HxCyan", hexColor("00FFFF"))

saveGifFile("c:\\aptilis\\colors.gif", b)

end if
```

Result:

For more details see the Bitmap topic.

Line

Aptilis 1

Line(BitmapHandle, x1, y1, x2, y2, Colour)

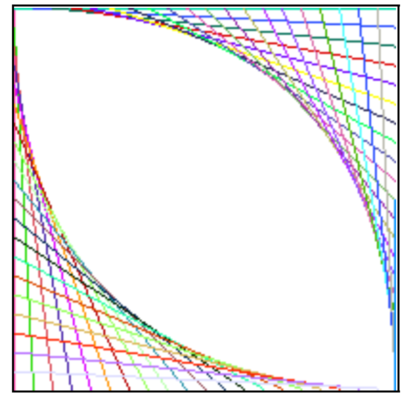
Line allows you to draw a line from (including) (x1, y1) to (including)(x2, y2).
The colour is specified through its index.

Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap.

Example:

```
b = createBitmap(200, 200)
if b = -1
print("Ooops, could not create the bitmap....\n")
else

// White background
clearBitmap(b, RGB(255, 255, 255))

for i=0 to 200 step 10

// We take random colours...
line(b, 0, i, i, 199, random(256))
line(b, i, 0, 199, i, random(256))

end for

saveGIFFile("lines.gif", b)

end if
```

Result:



For more details see the Bitmap topic.

OutputGifBitmap

Aptilis 1

OutputGifBitmap(BitmapHandle)

OutputGifBitmap is very similar to SaveGifFile except that the GIF picture is sent to the standard output, rather than to a file.
This comes very handy when you want to generate a graphic on the fly in a web page.
First, you need to put the IMG tag in your HTML document:

```
<img src="http://www.yourServer.com/cgi-bin/aptilis.exe?file=/home/aptilis/
          your_directory/yourAptilisProgram.e.txt&param1=12&param2=53" />
```

(You need the replace the different parts in the example above with values relevant to your server).
In the Aptilis script, before you send the picture, you need to send the appropriate magic line:

```
print("Content-type: image/gif\n\n")
```

And basically, that's all there is to it!

Return Value:
0, or –1 if the handle specified was incorrect. In case of file error, check the _errno variable for a more detailed explanation.

For more details see the Bitmap topic.

PrintAt

Aptilis 1

PrintAt(BitmapHandle, x, y, String, Colour)

PrintAt will write a string at a given position in previously created bitmap.
The coordinates given indicate the lower left corner of the rectangular zone in which the string is going to be written.

PrintAt uses a fixed–size font by default where all letters have a width of 8 pixel and a height of 16.
You can specify a font to use in a bitmap with setFont in which case it is difficult to estimate the length of a string, unless you call getStringMetrics.

Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap.

Example:

```
b = createBitmap(250, 50)
if b = -1
print("Ooops, could not create the bitmap....\n")
else

// White background
clearBitmap(b, RGB(255, 255, 255))

// Unix user need to find a true type font and copy its file
// somewhere on their system, and refer to it here correctly.
// Windows user can choose another ttf file if they want.

setFont(b, "c:\\windows\\system\\arib10.ttf", 20)

getStringMetrics(b, mf[], "Aptilis is ")
getStringMetrics(b, ms[], "COOL!")

// total width
w = mf[0] + ms[0]


x = (250 - w) / 2
// The y coordinate is the bottom of the string, hence the plus.
y = (50 + mf[1]) / 2


printAt(b, x, y, "Aptilis is", RGB(0, 0, 0))
printAt(b, x + mf[0], y, "COOL!", RGB(0, 255, 255))

saveGIFFile("font.gif", b)


end if
```

Result:

For more details see the Bitmap topic.

RGB

Aptilis 1

RGB(Red, Green, Blue)

RGB returns an Aptilis colour index from three values that represent the intensities of red, green and blue respectively for the requested colour.
These values should be in the range 0..255. Any value inferior to 0 will be considered 0, and any value bigger than 255 will be considered to be 255.
Aptilis provides you with 256 colours and the index returned matches the closest available colour. If the colour returned is not good enough, (You know that when you have seen the graphic), you can always override Aptilis'es colour definitions with setColor. But by doing so, you might prevent the font anti−aliasing to work properly.

Return Value:
An Aptilis colour index.

Example:

```
b = createBitmap(90, 200)
if b = -1
print("Oops...\n")
else

clearBitmap(b, RGB(255, 255, 255))


printAt(b, 4, 20, "Red", RGB(255, 0, 0))
printAt(b, 4, 40, "Green", RGB(0, 255, 0))
printAt(b, 4, 60, "Blue", RGB(0, 0, 255))
printAt(b, 4, 80, "Yellow", RGB(255, 255, 0))



printAt(b, 4, 100, "HxRed", hexColor("FF0000"))
printAt(b, 4, 120, "HxGreen", hexColor("00ff00"))
printAt(b, 4, 140, "HxBlue", hexColor("0000FF"))
printAt(b, 4, 160, "HxYellow", hexColor("FFFF00"))
printAt(b, 4, 180, "HxMagenta", hexColor("FF00FF"))
printAt(b, 4, 199, "HxCyan", hexColor("00FFFF"))

saveGifFile("c:\\aptilis\\colors.gif", b)

end if
```

Result:

For more details see the Bitmap topic.

SaveGifFile

Aptilis 1

SaveGifFile(FileName, BitmapHandle)

SaveGifFile saves a bitmap into a file, a bit like SaveFile does for more general purposes.
Of course, you will need to create and draw into a bitmap before you can use its handle to save it.
Once you're done with your bitmap, you can release the memory it's using by calling DeleteBitmap.

Return Value:
0, or –1 if the handle specified was incorrect.

For more details see the Bitmap topic.

SetBackground

Bitmaps

Aptilis 1

SetBackground(BitmapHandle, Colour)

setBackground allows you to specify a background color in order for the font anti−aliasing to work correctly. Note that the background colour is also redefined by clearBitmap, except that clearBitmap also repaints the whole bitmap.
Of course, if you have redefined some colours with setColor the anti−aliasing will probably not work as expected.
See also printAt.

Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap.

For more details see the Bitmap topic.

SetColor

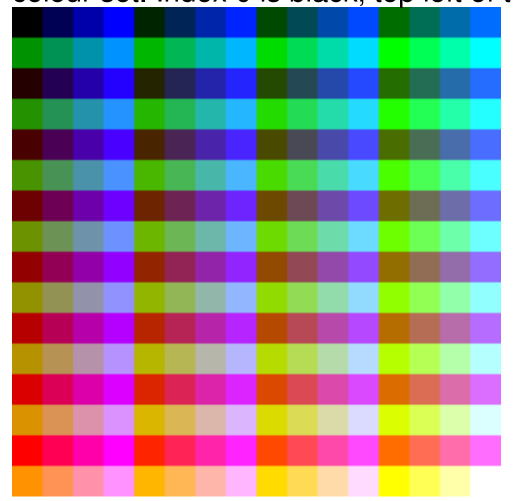
Aptilis 1

SetColor(BitmapHandle, ColourIndex, Red, Green, Value)

SetColor sets a color in a previously defined bitmap.
You can define up to 256 colours, and they range from 0 to 255.
You have to indicate a level a of red, a level of green and a level of blue.
Aptilis defines the colours for you when you create a bitmap. However, especially if you want to create pictures with a lot of close shades, you might not have all the colours you need in the default set, which is limited, and to be honest fairly poor in the blues.
By redefining colours however, you break the anti–aliasing scheme used by printAt.
Here is a reminder of how to mix colours:

r, v, b	Colour
0, 0, 0	Black
0, 0, 255	Blue
255, 0, 0	Green
0, 255, 0	Red
255, 0, 255	Magenta
0, 255, 255	Cyan
255, 255, 0	Yellow
255, 255, 255	White

For some, it might seem peculiar that, for example, green and red give yellow. This is because computer monitors (and TV screens) work according to additive synthesis, rather than the substractive variety, most commonly encountered with pigments and paper.
Just experiment, and you will soon be familiar with this 'weird' system. Here is the default Aptilis colour set. Index 0 is black, top left of the picture, 255 is white, bottom right.

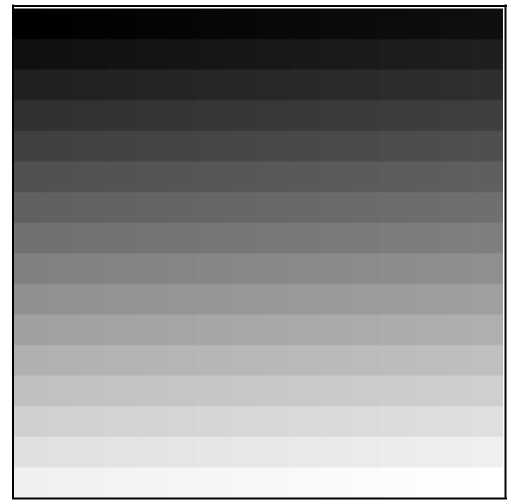


Return Value:
0 if everything was OK, or −1 if you specified an incorrect index or an incorrect bitmap.

Example:

```
b = createBitmap(256, 256)


// We want a 256 gray palette
if b = -1
print("Ooops, could not create the bitmap....\n")
else

// Gray values
for i=0 to 255
setColor(b, i, i, i, i)
end for

i = 0
for y=0 to 15
gy = y * 16
for x = 0 to 15
gx = x * 16
box(b, gx, gy, gx + 16, gy + 16, i, 1)
i = i + 1
end for
end for

saveGIFFile("grays.gif", b)

end if
```

Result:



For more details see the Bitmap topic.

SetFont

Bitmaps

Aptilis 1

SetFont(BitmapHandle, PathToFontFile[, FontSize])

SetFont attaches a font to a bitmap.
The default font is a bitmaped one, which can not be enlarged. All its letters have a width of 8 pixels and a height of 16.
Aptilis uses a True Type Library developped by David Turner, Robert Wilhelm and Werner Lemberg (See Link at bottom of page) which provides us with a very useful feature: we can use True Type fonts in bitmaps under Windows (NT/95) as well as under Unix!
Windows users can use the fonts usually stored in their \windows\system directory. Unix users will have to buy their True Type fonts or download some from the Internet, for example from 1001 Fonts, but make sure you download True Type fonts, and not, say, Adobe Type 1 fonts...
True Type fonts are vectorial, that means they can be enlarged without looking 'blocky'.
aptilis now integrates the FreeType 1.1 engine that produces great output. Aptilis might have some problems rendering fonts on couloured backgrounds, since it uses a limited palette for the sake of simplicity.
To revert to the system font, call setFont with an empty string as the font name:
setFont(bitmap, "")
Because an aptilis script has no way to know which kind of display it is going to be used with (every one connected to the net has got a different set−up), SetFont expects a font size in pixels.
It is much easier to understand (1 pixel = 1 point on the screen) but graphics may look tiny on high resolution screens or big on low resolution devices. That always was, and might still be for some time, a designer's challenge.

Determining the width of a string to be written with a True Type font is not as straightforward as calculating the width of a system string (you multiply the length of the string by 8), so there is a sub that does it for you: getStringMetrics.
Aptilis supports True Type fonts stored in ttf files, and currently does not implement font collections. (It takes the first font available in any case).

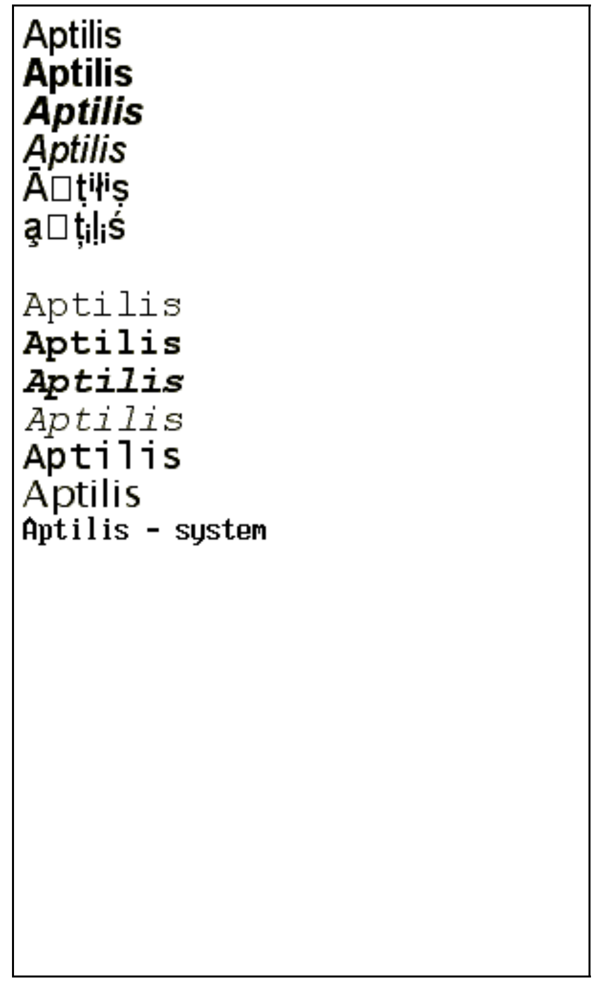
Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap, or an invalid font file or some other error happened. _errno will give details.

Example:

```
changeDirectory("C:\\windows\\system")
n = getFileList(fonts[], "*.ttf")



// Let's assume no font will have a height bigger than 20 pixels
height = (n + 1) * 22

print("n=", n, "\n")

b = createBitmap(300, height)
if b = -1
```

```
print("Oops, it was impossible to create a bitmap...\n")
else

white = RGB(255, 255, 255)
black = RGB(0, 0, 0)


clearBitmap(b, white)

for i=0 to 12
setFont(b, fonts[i]$, 20)
printAt(b, 5, (i + 1) * 20, "Aptilis", black)
end for

// back to the system font
setFont(b, "")
printAt(b, 5, (i + 1) * 20, "Aptilis - system", black)

saveGIFFile("c:\\aptilis\\setfont.gif", b)

deleteBitmap(b)
end if
```

Result:



For more details see the Bitmap topic.
See also:
The Freetype Home Page.

SetPixel

Bitmaps

Aptilis 1

SetPixel(BitmapHandle, x, y, Colour)

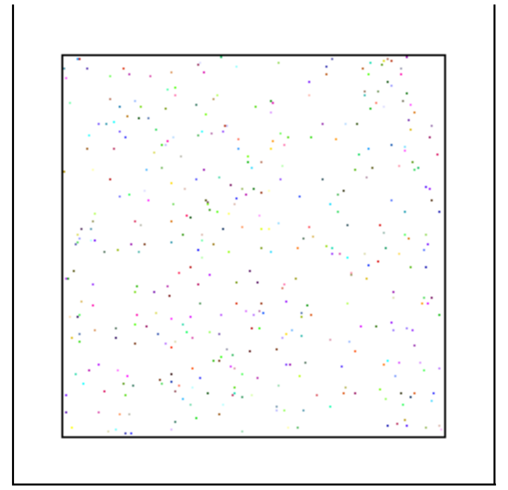
SetPixel is the most basic graphic instruction that allows you to set a single pixel (or point) in a previously created bitmap.
You indicate a colour by its index, which you get from functions like RGB or hexColor or you can pick an index and define its colour value with SetColor.

Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap.

Example:

```
b = createBitmap(250, 250)
if b = -1
print("Oops, it was impossible to create a bitmap...\n")
else

white = RGB(255, 255, 255)
black = RGB(0, 0, 0)

clearBitmap(b, white)

box(b, 25, 25, 225, 225, black, 0)

// Random points: random position, random colours
for i=0 to 400

setPixel(b, 26 + random(198), 26 + random(198), random(255))

end for

saveGifFile("points.gif", b)

deleteBitmap(b)
end if
```

Result:



For more details see the Bitmap topic.

SetThickness

Bitmaps

Aptilis 1

SetThickness(BitmapHandle, Thickness)

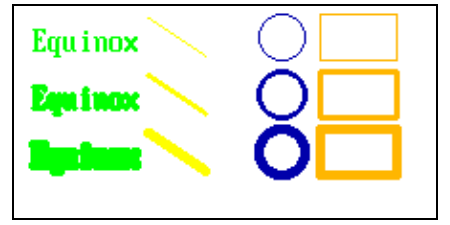
SetThickness allows you to specify how thick points should be when drawing pixels, lines, ellipses, boxes, and text.
The only acceptable values are: 1, 3 and 5. They set drawn points diameters to respectively 1, 3 and 5 pixels. Text written in thickness 5 is hardly legible, where 3 gives text a 'bold' aspect.
True Type Fonts selected through setFont are not affected by setThickness, only the bitmaped, default font is.

Return Value:
0 if everything was OK or −1 if you indicated a non−valid bitmap or an incorrect thickness.

Example:

```
b = createBitmap(220, 110)
if b = -1
print("Oops, it was impossible to create a bitmap...\n")
else

white = RGB(255, 255, 255)
clearBitmap(b, white)

// Thickness valid values are 1, 3, 5
for i=1 to 5 step 2

thickDemo(b, i)

end for

saveGIFFile("thick.gif", b)

deleteBitmap(b)
end if


sub thickDemo(bitmap, thickness)

y = thickness * 15

setThickness(bitmap, thickness)
printAt(bitmap, 10, y + 10, "Aptilis", RGB(0, 255, 0))
line(bitmap, 70, y-10, 100, y + 10, RGB(255, 255, 0))
ellipse(bitmap, 140, y, 12, 12, RGB(0, 0, 190))
box(bitmap, 160, y-12, 200, y + 12, RGB(255, 190, 0), 0)


end thickDemo
```

Result:



For more details see the Bitmap topic.

Databases

- AppendRecord
- DeleteRecord
- GetAllFields
- GetAllRecordsByKey
- GetAllRecordsByNearKey
- GetField
- GetFixedLengthField
- GetRecordIndexByKey
- GetRecordIndexByNearKey
- LoadDatabase
- MakeFixedLengthField
- MakeRecord
- ParseDatabase
- SaveDatabase
- SortDatabase

AppendRecord

Databases

Aptilis 1

AppendRecord(FileName, Record)

AppendRecord allows you to add a single record at the end of an existing database.
While in variables, records are stored in a platform dependant, 'weird' format, that is not readily understandable by humans.
AppendRecord and saveDatabase convert this machine−friendly format into plain text that makes sense for us.

Return value:
The number of characters written, or −1 in case of error.

Example:

```
// just to be tidy
clearArray(r[])

// Animal, How many legs, feeds on
r[0] = makeRecord("Dog", "4", "Meat, old shoes")
r[1] = makeRecord("Cat", "4", "Jimmy the goldfish")
r[2] = makeRecord("Man", "2", "What's on the menu?")

saveDatabase("my_db.txt", r[])

// Oops! We forgot one! (The record could be added at any time)
forgottenRecord = makeRecord("Shark", "0", "People")
appendRecord("my_db.txt", forgottenRecord$)
```

The file 'my_db.txt' now contains:

```
"Dog","4","Meat, old shoes"
"Cat","4","Jimmy the goldfish"
"Man","2","What's on the menu?"
"Shark","0","People"
```

DeleteRecord

Databases

Aptilis 1

DeleteRecord(Database[], Key, Position, n[, startFrom])

DeleteRecord will scan a database and remove each record that has a field matching the given Key at the given Position. In database parlance, the Position of the key is it's column.
It will remove up to 'n' records or as many it finds that match the request. If you specify a value of 1 for n, then only the first matching record will be erased.
You can use the extra 'startFrom' parameter to skip the first 'startFrom' records of the database.

Return value:
The number of records removed.

Example 1:
We remove all matching records. "mammal" will be found in column '3', the French name has an index of '0' and so on.

```
sub main


db[0] = makeRecord("chien","dog","1","mammal") $
db[1] = makeRecord("chat","cat","2","mammal") $
db[2] = makeRecord("poisson","fish","3","vertebrate") $
db[3] = makeRecord("escargot","snail","4","invertebrate") $
db[4] = makeRecord("souris","mouse","5","mammal") $
db[5] = makeRecord("perroquet","parrot","6","bird") $
db[6] = makeRecord("dauphin","dolphin","7","mammal") $
db[7] = makeRecord("fleur","flower","8","plant") $

n = getArraySize(db[])
deleteRecord(db[], "mammal", 3, n)

n = getArraySize(db[])
print("New size: ", n, "\n")
for i=0 to n-1
getAllFields(fl[], db[i]$)
print(fl[1]$, " ", fl[3]$, "\n")
end for


end sub
```

Result:

```
New size: 4.000000
fish vertebrate
snail invertebrate
parrot bird
flower plant
```

Example 2:
Here, we specify to remove only one field, so only poor doggy will go. :-(

```
sub main
```

```
db[0] = makeRecord("chien","dog","1","mammal") $
db[1] = makeRecord("chat","cat","2","mammal") $
db[2] = makeRecord("poisson","fish","3","vertebrate") $
db[3] = makeRecord("escargot","snail","4","invertebrate") $
db[4] = makeRecord("souris","mouse","5","mammal") $
db[5] = makeRecord("perroquet","parrot","6","bird") $
db[6] = makeRecord("dauphin","dolphin","7","mammal") $
db[7] = makeRecord("fleur","flower","8","plant") $


n = getArraySize(db[])
deleteRecord(db[], "mammal", 3, 1)

n = getArraySize(db[])
print("New size: ", n, "\n")
for i=0 to n-1
getAllFields(fl[], db[i]$)
print(fl[1]$, " ", fl[3]$, "\n")
end for

end sub
```

Result:

```
New size: 7.000000
cat mammal
fish vertebrate
snail invertebrate
mouse mammal
parrot bird
dolphin mammal
flower plant
```

Example 3:

Lastly, we specify to remove as many records as possible, however, we start at positon 2, hence saving both dog and cat from oblivion.

```
sub main

db[0] = makeRecord("chien","dog","1","mammal") $
db[1] = makeRecord("chat","cat","2","mammal") $
db[2] = makeRecord("poisson","fish","3","vertebrate") $
db[3] = makeRecord("escargot","snail","4","invertebrate") $
db[4] = makeRecord("souris","mouse","5","mammal") $
db[5] = makeRecord("perroquet","parrot","6","bird") $
db[6] = makeRecord("dauphin","dolphin","7","mammal") $
db[7] = makeRecord("fleur","flower","8","plant") $


n = getArraySize(db[])
deleteRecord(db[], "mammal", 3, n, 2)

n = getArraySize(db[])
print("New size: ", n, "\n")
for i=0 to n-1
getAllFields(fl[], db[i]$)
print(fl[1]$, " ", fl[3]$, "\n")
```

```
end for

end sub
```

Result:

```
New size: 6.000000
dog mammal
cat mammal
fish vertebrate
snail invertebrate
parrot bird
flower plant
```

GetAllFields

Databases

Aptilis 1

GetAllFields(DestArray[], Record)

GetAllFields loads all the fields of a record straight into an array. This saves you from having to do it for each field individually.

Return value:
The number of fields found. That can be zero.

Example:
We are going to load the following database, called 'db.txt':

```
"Mouse","mammal","cheese"
"Horse","mammal","Hay, grass"
"Fish","vertebrate","alguae, other fish, plancton"
"Snail","invertebrate","lettuce"
"Termite","insect","your house"
"Man","mammal","meat, vegetables, fruits"
"Leelou","furry thing","cat food in cans"
```

Now the code:

```
n = loadDatabase(db[], "db.txt")
for i=0 to n-1
getAllFields(fields[], db[i]$)
print("Name: ", fields[0]$, "\n")
print("Type: ", fields[1]$, "\n")
print("Feeds on: ", fields[2]$, "\n")
print("++++++\n")
end for
```

Result:

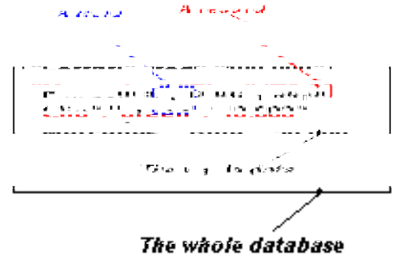
```
Name: Mouse
Type: mammal
Feeds on: cheese
++++++
Name: Horse
Type: mammal
Feeds on: Hay, grass
++++++
Name: Fish
Type: vertebrate
Feeds on: alguae, other fish, plancton
++++++
Name: Snail
Type: invertebrate
Feeds on: lettuce
++++++
Name: Termite
Type: insect
Feeds on: your house
++++++
```

```
Name: Man
Type: mammal
Feeds on: meat, vegetables, fruits
++++++
Name: Leelou
Type: furry thing
Feeds on: cat food in cans
++++++
```

(I apologize to the biologist community for my approximative knowledge of living things. Leelou is my lovely kitty.)

See also:
loadDatabase, getField.

Database speak:



GetAllRecordsByKey

Databases

Aptilis 1

GetAllRecordsByKey(dbDest[], dbSrc[], key, keypos)

Get**All**RecordsByKey will extract records from a database (dbSrc) into a new one (dbDest) according to a criteria specified by you.
This criteria is a field specified by 'key' in the column 'keypos'.
If you want to do a search using wild cards (like '*' or '?') to do approximative matches, use GetAllRecordsBy**Near**Key.

You read and write databases from/to files with loadDatabase and saveDatabase respectively.

GetRecordIndexByKey and GetRecordIndexByNearKey are more low−level and only retrieve *the index* of one record at a time.

Return value:
The number of records(s) retrieved.

Example

```
sub main

db[0] = makeRecord("chien","dog","1","mammal") $
db[1] = makeRecord("chat","cat","2","mammal") $
db[2] = makeRecord("poisson","fish","3","vertebrate") $
db[3] = makeRecord("escargot","snail","4","invertebrate") $
db[4] = makeRecord("souris","mouse","5","mammal") $
db[5] = makeRecord("perroquet","parrot","6","bird") $
db[6] = makeRecord("dauphin","dolphin","7","mammal") $
db[7] = makeRecord("fleur","flower","8","plant") $


n = GetAllRecordsByKey(newDb[], db[], "mammal", 3)
print("Found: ", n, " records\n")

for i=0 to n-1
getAllFields(fl[], newDb[i]$)
print(fl[1]$, " ", fl[3]$, "\n")
end for
end main
```

Result:

```
Found: 4.000000 records
dog mammal
cat mammal
mouse mammal
dolphin mammal
```

See also getField and getAllFields
Notes

- Spaces and case
  getAllRecordsByKey **IS** case sensitive.
  Also, it will take into account any space you type. So be careful with " blue" and "blue" which are as different as chalk and cheese.
- Format of databases
  The format of the database is left to you as long as you keep the double quotes and the commas where they 're supposed to be. **You cannot have a double quote within a field**. Then it's up to you to arrange the fields in any order you want. **You can have up to 256 fields** and they can be as long as you want.
- There is a limited support for fixed length field databases through getFixedLengthField and makeFixedLengthField.

GetAllRecordsByNearKey

Databases

Aptilis 1

GetAllRecordsByNearKey(dbDest[], dbSrc[], key, keypos)

Get**All**RecordsBy**Near**Key will extract records from a database (dbSrc) into a new one (dbDest) according to a criteria specified by you.
This criteria is a field specified by 'key' in the column 'keypos'.
Because it's a 'near' search, the criteria doesn't need to match a field precisely for a record to be copied into the new database.
For example, if you do a search on "bl*" for a field containing the colour of a car, then you will get all cars which colours are:
"Blue", "Black". (I can't think of any other colour starting with 'bl')
GetAllRecordsByNearKey is **not** case sensitive.

You read and write databases from/to files with loadDatabase and saveDatabase respectively.

If you want to do an exact search use GetAllRecordsByKey.

GetRecordIndexByKey and GetRecordIndexByNearKey are more low−level and only retrieve *the index* of one record at a time.

Return value:
The number of record(s) retrieved.

Example
Retrieving the creepy−crawlies...

```
sub main

db[0] = makeRecord("chien","dog","1","mammal") $
db[1] = makeRecord("chat","cat","2","mammal") $
db[2] = makeRecord("poisson","fish","3","vertebrate") $
db[3] = makeRecord("escargot","snail","4","invertebrate") $
db[4] = makeRecord("souris","mouse","5","mammal") $
db[5] = makeRecord("perroquet","parrot","6","bird") $
db[6] = makeRecord("dauphin","dolphin","7","mammal") $
db[7] = makeRecord("fleur","flower","8","plant") $


n = GetAllRecordsByNearKey(newDb[], db[], "*brate", 3)
print("Found: ", n, " records\n")

for i=0 to n-1
getAllFields(fl[], newDb[i]$)
print(fl[1]$, " ", fl[3]$, "\n")
end for

end main
```

Result:

```
Found: 2.000000 records
fish vertebrate
snail invertebrate
```

See also getField and getAllFields

Notes

Examples

To do an aproximate search, use the special characters '?' for any one letter, and '*' for any group of letter.

"blue"	will match 'blue', exactly.
"f*"	All fields begining with 'f'.
"*i"	All fields ending with 'i'.
"*a*"	All fields containing the letter 'a'.
"?ree"	All words of any first letter ending in 'ree', like 'tree' or 'free'.
"*a"	Anything ending with the letter a, like 'ha' or 'armada'
etc...	

The patterns used by 'GetAllRecordsByNearKey' are exactly the same as the ones used by Match.
There is a limited support for fixed length field databases through getFixedLengthField and makeFixedLengthField.

GetField

Databases

Aptilis 1

GetField(Record, Index)

GetField will isolate a field in a database record.
Databases are loaded with loadDatabase or are made up of records put in arrays with makeRecord.
Records can also be found with functions like getrecordindexbykey and getrecordindexbynearkey.
You can also load all the fields of a record in one go into an array with getAllFields.

Return value:
The field requested, or an empty string if the index goes beyond the number of fields available.

Example:

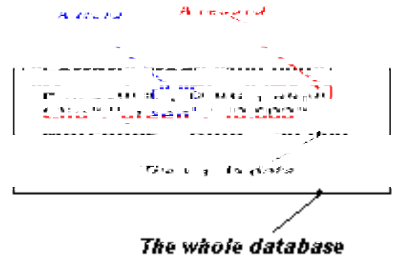
```
r = makeRecord("zero","one","two","three") $
print(getfield(r$, 1)$)
```

Result:

```
one
```

See also:
getAllFields getFixedLengthField

A database:



GetFixedLengthField

Databases

Aptilis 1

GetFixedLengthField(String, Index, length_of_field)

GetFixedLengthField will isolate a field in a record of which each field has a known length. It works exactly like mid except that GetFixedLengthField will remove all the trailing spaces and tab characters at the end of the field.

Return value:
A substring from another string with terminating spaces and tabs trimmed off.

Example:

```
// first field: 12 chars, second field: 20 chars, third field 30 chars
// All padded with spaces

flrecord = "car         Volvo               Sweden                        " $

print("[", getFixedLengthField(flrecord$, 1, 12)$, "]\n")
print("[", getFixedLengthField(flrecord$, 13, 20)$, "]\n")
print("[", getFixedLengthField(flrecord$, 33, 30)$, "]\n")
```

Result:

```
[car]
[Volvo]
[Sweden]
```

See also:
getField, the equivalent native Aptilis format sub.

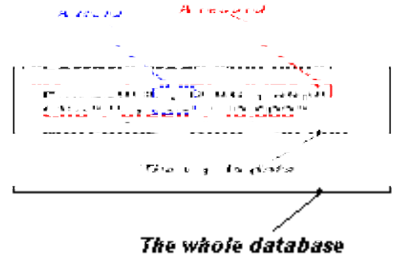
Notes:
The fixed length field databases is not the native Aptilis database format. (see LoadDataBase). However, MakeFixedLengthField and getFixedLengthField allow you to load/save databases using that format.

GetRecordIndexByKey

Aptilis 1

GetRecordIndexByKey(ArrayName[], Key, Index [, FromWhere])

GetRecordIndexByKey scans a database that has usually been loaded with loadDatabase.
The function returns an index, which you can then use to retrieve a record.



The *Key* you indicate will be compared to all the fields at the *Index*th position in each record. Field indexes start at 0.
To search for multiple matches, you can indicate from which record you want to start. That's the optional *FromWhere* parameter. The first record also has an index of 0. You must start at 0, and then set the start parameter to (the found index + 1) or stop the search if no match was found.
If no match is found, the function returns −1.
A approximative search can also be done with getRecordIndexByNearKey.

Get**All**RecordsByKey and Get**All**RecordsByNearKey are more high−level and allow you to retrieve several records directly, in one go.

Return value:
A valid index in the database, −1 otherwise.

Example 1:
a database

```
"Ferrari","red","100,000","2 seats"
"Mercedes","Blue","50,000","2 seats and a half"
"Clio","green","10,000","Nearly five seats"
"Jaguar","blue","200,000","9 seats"
"Lotus","Silver","250,000","2 seats"
```

Name of car: field 0
Colour: field 1
Price: field 2
Seats: field 3
We are going to load the database with the loaddatabase function.
This one is made up of 5 records, each with four fields.

Example 2:
Assuming we are on a Windows type machine...

```
// Load the database into, say, 'db'
```

```
// We won't use n in this example
n = loadDatabase(db[], "c:\\databases\\cars\\cars.txt")

// We want to find the record that says 'colour blue'
// The colours are stored in Field 1.
ix = getRecordIndexByKey(db[], "blue", 1)

if ix = -1
print("No record found\n")
else
// First, let's print the index
print("Record found at: ", int(index), "\n")

// Then, just the name of the car
n = getField(db[ix]$, 0) $

print("The ", n$, " is blue\n")

end if
```

Result:

```
"Record found at: 1
The Mercedes is blue
```

See also getField and getAllFields

Example 3:
Here, we search for all the records.

```
loadDatabase(db[], "c:\\databases\\cars\\cars.txt")

ix = 0
repeat
ix = getRecordIndexByKey(db[], "blue", 1, ix)
if ix <> -1

// Just the name of the car
n = getField(db[ix]$, 0) $

print("The ", n$, " is blue\n")

// Start at the next record, otherwise we would loop indefinetly
ix = ix + 1

end if

until ix = -1
```

Result:

```
The Mercedes is blue
The Jaguar is blue
```

Notes

- Spaces and case
  getRecordIndexByKey **IS** case sensitive.

(Contrary to what was said here before)
Also, it will take into account any space you type. So be careful with " blue" and "blue" which are as different as chalk and cheese.

- Format of databases
  The format of the database is left to you as long as you keep the double quotes and the commas where they're supposed to be. **You cannot have a double quote within a field**. Then it's up to you to arrange the fields in any order you want. **You can have up to 256 fields** and they can be as long as you want.
- There is a limited support for fixed length field databases through getFixedLengthField and makeFixedLengthField.

GetRecordIndexByNearKey

Databases

Aptilis 1

GetRecordIndexByNearKey(ArrayName[], Key, Index [, FromWhere])

GetRecordIndexBy**Near**Key works exactly like getRecordIndexByKey except that it does an approximative search.
Here, I will just explain how to do the keys.
See getRecordIndexByKey for programming examples.
getRecordIndexByNearKey is **not** case sensitive.

Get**All**RecordsByKey and Get**All**RecordsByNearKey are more high−level and allow you to retrieve several records directly, in one go.

Examples
To do an aproximate search, use the special characters '?' for any one letter, and '*' for any group of letter.

"blue"	will match 'blue', exactly.
"f*"	All fields begining with 'f'.
"*i"	All fields ending with 'i'.
"*a*"	All fields containing the letter 'a'.
"?ree"	All words of any first letter ending in 'ree', like 'tree' or 'free'.
"*a"	Anything ending with the letter a, like 'ha' or 'armada'
etc...	

Notes
The patterns used by 'GetRecordIndexByNearKey' are exactly the same as the ones used by Match.

LoadDatabase

Databases

Aptilis 1

LoadDatabase(DestinationArrayName[], FileName[, login, password])

LoadDatabase will load a database from a file and will put it into an array.
If you want to parse a database yourself from a string, have a look at ParseDatabase.

LoadDatabase does not need the login and password parameters, unless you are trying to load the database over the ftp protocol:
loadDatabase(db[], "ftp://ftp.server.com/somedir/anotherdir/aDatabase.txt", "login", "password")
Protocols supported: ftp and file.

The function returns the number of records loaded, or −1 if there has been an error, in which case you can have a look at _errno$ for details about what went wrong with the file.

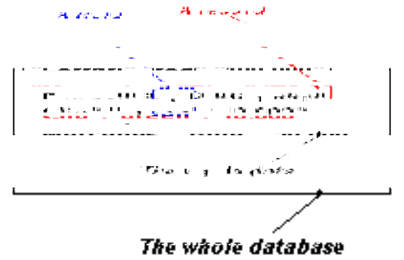
IMPORTANT
Any parsing error will cause loading to stop.

The database must be a text file with one record per line, the lines being separated by either a LineFeed (LF), or a a Carriage Return plus a LineFeed(CR/LF).
Each field should be enclosed in double quotes and separated from the next field by a comma. (See example 1) **You cannot have a double quote within a field**
Database files can be created with a simple text editor or exported from commercial database packages like Access, Paradox, etc...

From within an Aptilis programm, to create a database, you have to create records with makeRecord. Although they are printable as strings, database records are not directly usable. The way they are stored internally will vary from platform to platform, and you cannot safely make assumptions about how a record is stored. (For example, integers may vary in size, as well as the way bits are organized within them)
You must use getField or getAllFields to retrieve fields.
If you are concerned a large database you loaded is hampering performances, you can free up the resources it's using, −once you are done with the database− by using clearArray on the array it's been loaded into.

Return value:
The number of records in the database or −1 in case of error, and _errno has some details.

Example 1:
a database:



What loaddatabase does is to load each line (=record) of the database in an element of the array you have specified.

Example 2:
Loading the above database, assuming we are on a windows type machine.

```
// Backslash is a special character!
n = loaddatabase(db[], "c:\\databases\\cars\\cars.txt")
print("There are ", int(n), " records")
```

```
There are 3 records
```

There are other database functions that allow you to isolate one or all the fields from a record, (getField or getAllFields) and to search a database that has been loaded for a given record, according to one field (getRecordIndexByKey, getRecordIndexByNearKey).
Use sortDatabase to sort databases.

You can also create a database within Aptilis by putting records in an array, with makeRecord and then save the database with saveDatabase

MakeFixedLengthField

Databases

Aptilis 1

MakeFixedLengthField(Content, Length[, Padding Character])

MakeFixedLengthField creates a string from the string you gave it, and if necessary it will add spaces so that the new string will contain the specified number of characters.
If the original string contains more characters than requested, the new string will contain a truncated version of the original string.
You can optionally specify a different padding character.

Return value:
A string, possibly padded with strings, of the required length.

Example:

```
flf1 = makeFixedLengthField("Hello", 10) $
flf2 = makeFixedLengthField("Hello", 12, ".") $
flf3 = makeFixedLengthField("Good morning", 4, ".") $

print("len = ", int(len(flf1$)), " [", flf1$, "]\n")
print("len = ", int(len(flf2$)), " [", flf2$, "]\n")
print("len = ", int(len(flf3$)), " [", flf3$, "]\n")
```

Result:

```
len = 10 [Hello     ]
len = 12 [Hello.......]
len = 4 [Good]
```

Notes:
The fixed length field databases is not the native Aptilis database format. (see LoadDataBase).
However, MakeFixedLengthField and getFixedLengthField allow you to load/save databases using that format.

MakeRecord

Databases

Aptilis 1

MakeRecord(Field1, Field2, Field3....)

MakeRecord allows you to create an individual database record, and by storing several records sequentially in an array, you can build up a database.
You need to use MakeRecord to create records in order to have them in the Aptilis specific fashion, so that you can use saveDatabase, getField and getAllFields.
Those functions implement and use the internal Aptilis database format which speeds up field extraction.
Although they are printable as strings, database records are not directly usable. The way they are stored internally will vary from platform to platform, and you cannot safely make assumptions about the way a record is stored. (For example, integers may vary in size, as well as the way bits are organized within them)

Return value:
A string where the record is stored in a special format.

Example:

```
// fields can be made up of strings
db[0] = makeRecord("Orange", "Orange", "Sweet, acid") $
db[1] = makeRecord("Banana", "Yellow", "Sweet") $
db[2] = makeRecord("Strawberry", "Red", "Sweet, subtle") $



// They can be made up of a mix of local and global variables
f1 = "Mango" $
_f2 = "Orange" $
f[2] = "Bitter" $

db[3] = makeRecord(f1$, _f2$, f[2]$) $



// They can also be made from an array...
fields[0] = "Lemon" $
fields[1] = "Yellow" $
fields[2] = "Acid" $

db[4] = makeRecord(fields[]) $



// or any combination of different type of values.

_nf[0] = "green" $
_nf[1] = "sweet, acid" $
db[5] = makeRecord("kiwi", _nf[]) $
```

```
//Finally, we save the file...
saveDatabase("c:\\aptilis\\db3.txt", db[])
```

The file db3.txt now contains:

```
"Orange","Orange","Sweet, acid"
"Banana","Yellow","Sweet"
"Strawberry","Red","Sweet, subtle"
"Mango","Orange","Bitter"
"Lemon","Yellow","Acid"
"kiwi","green","sweet, acid"
```

ParseDatabase

Aptilis 2

ParseDatabase(db[], dababase_in_a_string)

ParseDatabase will parse a database from a string and will put it into an array.
If you want to load a database from a file, have a look at LoadDatabase.

The function returns the number of records loaded.

IMPORTANT
Any parsing error will cause parsing to stop.

The database must be text with one record per line, the lines being separated by either a LineFeed (LF), or a a Carriage Return plus a LineFeed(CR/LF).
Each field should be enclosed in double quotes and separated from the next field by a comma. (See example 1) **You cannot have a double quote within a field**
Database files can be created with a simple text editor or exported from commercial database packages like Access, Paradox, etc...

From within an Aptilis programm, to create a database, you have to create records with makeRecord. Although they are printable as strings, database records are not directly usable. The way they are stored internally will vary from platform to platform, and you cannot safely make assumptions about the way a record is stored. (For example, integers may vary in size, as well as the way bits are organized within them). The special format that databases are stored in allows quick retrieval of information by Aptilis.
You must use getField or getAllFields to retrieve fields.
If you are concerned a large database you are using is hampering performances, you can free up the resources it's using, −once you are done with the database− by using clearArray on the array it's been parsed into.

Return value:
The number of records in the database or −1 in case of error, and _errno has some details.

There are other database functions that allow you to isolate one or all the fields from a record, (getField or getAllFields) and to search a database that has been loaded for a given record, according to one field (getRecordIndexByKey, getRecordIndexByNearKey).
Use sortDatabase to sort databases.

You can also create a database within Aptilis by putting records in an array, with makeRecord and then save the database with saveDatabase

SaveDatabase

Databases

Aptilis 1

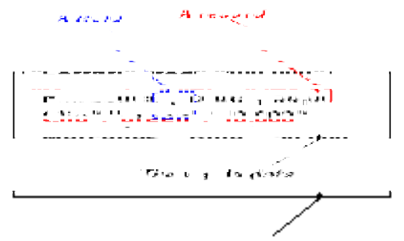
SaveDatabase(FileName, DatabaseArray[][, login, password])

SaveDatabase is the reverse of loadDatabase which loads a text database file into an array.
Each record is an element in the array, however the fields are not directly readable because they are stored in a format that speeds up their extraction.
If you have added some new records to a database (with makeRecord) or created a database from scratch, and want to save it to the usual text format, then you should use saveDatabase.

SaveDatabase can save a database to another PC by using the FTP protocol. This is when you need to specify a login and password.
saveDatabase("ftp://ftp.server.com/somedir/anotherdir/aDatabase.txt", db[], "login", "password")
Protocols supported: ftp and file.

Return value:
the total number of bytes written, or −1 if there has been an error in which case _errno might contain details of the problem.

See makeRecord for a programing example.



SortDatabase

Aptilis 2

SortDatabase(db[],WhatField,sortType)

SortDatabase(db[], whatField, sortType) will sort a database using one of its fields as a sorting criteria.
Indicate the field to sort by in WhatField, 0 being the first one, 1, the second one, etc.
Sorting can be done alphabetically or numerically by setting 'sortType' to either "alphabetical" or "numeric".
To sort in reverse order, sort normally, and then use reverseArray.

Use sortArrays on normal arrays arrays (not databases).

Return value:
0 if everything was OK or −1 if there wasn't enough memory to carry out the operation.

Example:

```
sub splash(title)


print(title$, "\n", string(len(title$), "-")$, "\n")

n = getArraySize(_animals[])
for i=0 to n - 1

getAllFields(fls[], _animals[i]$)
print(join(fls[], " * ")$, "\n")

end for


print("\n")

end splash


sub main()


loadDatabase(_animals[], "animals.txt")


splash("Rough")


sortDatabase(_animals[], 2, "numeric")
splash("Numeric")


sortDatabase(_animals[], 0, "alphabetical")
splash("Alpha->by English names")
```

```
sortDatabase(_animals[], 1, "alphabetical")
splash("Alpha->by French names")



end main
```

Result:

```
Rough
-----
Dog * Chien * 14 * mammal * shoes
Cat * Chat * 3 * flying mammal * flies
Freddy the Goldfish * Fred Le Poisson rouge * 0.5 * fish * canned fish food
Teebo * Thibault * 29 * apparantly mammal * pizza
Bird * Oiseau * 2 * Dinosaur * insects

Numeric
-------
Freddy the Goldfish * Fred Le Poisson rouge * 0.5 * fish * canned fish food
Bird * Oiseau * 2 * Dinosaur * insects
Cat * Chat * 3 * flying mammal * flies
Dog * Chien * 14 * mammal * shoes
Teebo * Thibault * 29 * apparantly mammal * pizza

Alpha->by English names
-----------------------
Bird * Oiseau * 2 * Dinosaur * insects
Cat * Chat * 3 * flying mammal * flies
Dog * Chien * 14 * mammal * shoes
Freddy the Goldfish * Fred Le Poisson rouge * 0.5 * fish * canned fish food
Teebo * Thibault * 29 * apparantly mammal * pizza

Alpha->by French names
----------------------
Cat * Chat * 3 * flying mammal * flies
Dog * Chien * 14 * mammal * shoes
Freddy the Goldfish * Fred Le Poisson rouge * 0.5 * fish * canned fish food
Bird * Oiseau * 2 * Dinosaur * insects
Teebo * Thibault * 29 * apparantly mammal * pizza
```

There are other database functions that allow you to isolate one or all the fields from a record, (getField or getAllFields) and to search a database that has been loaded for a given record, according to one field (getRecordIndexByKey, getRecordIndexByNearKey).

Files

AppendToFile

Files

Aptilis 1

AppendToFile(FileName, Content)

AppendToFile will add a string at the end of a file.
If the file did not exist, it is created.
The path indicated should be constructed according to your platform.
Windows/DOS way: c:\texts\accounts\year98.txt
Unix way: /texts/accounts/year98.txt
Note the different slashes...

AppendToFile does not understand the ftp:// type, but SavFile (in the internet family) does.

Return Value:
Number of bytes written.
If the value is different from the length of the string you specified, then check the _errno variable for details.

Example:
The file *myfile.txt* aready contains:

```
Hello world
-----------
```

With the code:

```
appendToFile("myfile.txt","\nthird line")
```

The file now contains:

```
Hello world
-----------
third line
```

See also loadfile, savefile.

Notes
Using AppendToFile heavily is not recommended when good performances are required.
Rather, it is better to work on a variable, append content to it and then use SaveFile.

ChangeDirectory

Files

Aptilis 1

ChangeDirectory(DirectoryName)

ChangeDirectory changes the directory your programme is running in to the one you specify.
Changing directories allows you to access different files stored in various places. Alternatively, you can indicate the full path of a file each time, but this might not be as efficient.
The path indicated should be constructed according to your platform.
Windows/DOS way: c:\texts\accounts
Unix way: /texts/accounts
(Don't forget that '\' is a special character and must be writen '//' in strings)

Return Value:
0 if everything was OK.
–1 in case of error and the _errno variable contains more details about the error.

Example:

```
d = getCurrentDirectory() $
print(d$, "\n")

// There are two backslashes, because 'backslash' is a special character
changedirectory("c:\\windows")

d = getCurrentDirectory() $
print(d$)
```

Result:

```
c:\aptilis
c:\windows
```

See also GetCurrentDirectory.

CreateDirectory

Aptilis 1

CreateDirectory(NewDirectoryName)

CreateDirectory will create a new directory in the current directory. You can check which directory your programme is running in with getCurrentDirectory and you can get a list of the existing directories with getDirectoryList.

Return Value:
0 if everything was OK.
–1 in case of error and errno contains more details about the error.

DeleteDirectory

Files

Aptilis 1

DeleteDirectory(DirectoryName)

DeleteDirectory will delete the directory you specified.
The directory you want to delete must be empty, otherwise the deletion will fail.
Use getFileList, getDirectoryList and deleteFile to check and if necessary delete the content of a directory.

Return Value:
0 if everything was OK.
−1 in case of error and the _errno variable contains more details about the error.

DeleteFile

Aptilis 1

DeleteFile(FileName)

DeleteFile erases a file and removes its name from the directory where it was.
You have to be careful with this function, because, especially under Unix, a file deleted is a file lost... forever!

A locked file you call delete upon, will be unlocked unconditinally (even if you locked it several times) and then deleted. In effect, no only do you delete the file, but you also allow other processes to get a lock on that file name.

Return Value:
0 if everything was OK.
–1 in case of error and the _errno variable contains more details about the error.

See also Lock and Unlock.

FileExist

Aptilis 1

FileExist(FileName)

FileExist allows you to check the existence of a file.

Return Value:
0 if the file exists, otherwise –1.

GetCurrentDirectory

Files

Aptilis 1

GetCurrentDirectory()

getcurrentdirectory returns the current directory your programme is running in.

Return Value:
A string containing the current directory
If the string is empty (its length being 0) then you might want to check the _errno variable for a possible error.

Example:

```
d = getcurrentdirectory() $
print(d$)
```

Result:

```
c:\aptilis
```

See also changeDirectory.

GetDirectoryList

Files

Aptilis 1

GetDirectoryList(DestinationArrayName[][, Filter])

getDirectoryList will fill the specified array with all the sub–directory names of the current directory.
The optional filter allows to get a list of directories restricted to, for example, directories that start with a given letter. The rules that apply to the filter are the same as the ones that apply to match and this is an aptilis implementation, not a system one, so that the behaviour of the filter functions is identical across the various platforms Aptilis runs on.
Note that both '.' and '..' are returned.
You can check which directory you're in with getCurrentDirectory and change to another directory with changeDirectory.
The array specified is emptied before the directories may be searched.

Return Value:
The number of directories found and the directories names in the specified array.

Example:

```
chdir("c:\\windows")
n = getdirectorylist(dl[])

for i=0 to n - 1

print(dl[i]$, "\n")

end for
```

Result:

```
.
..
system
command
config
system32
```

See also: getFileList, getCurrentDirectory, changeDirectory.

GetFileDate

Files

Aptilis 1

GetFileDate(FileName)

GetFileDate returns the date of creation of the file you specified.
The value returned is the number of seconds elapsed since January 1, 1970.
Such functions as fillTimeArray and fillLocalTimeArray allow you to transform this value into more explicit values for days, months and years.

Return Value:
The date of creation if everything was OK.
–1 in case of error and the _errno variable contains more details about the error.

Example:

```
f = "c:\\windows\\tjmail.exe" $
t = getFileDate(f$)
fillLocalTimeArray(time[], t)
print(int(time[3])$, "/", int(time[4]+1)$, "/" ,int(time[5])$, "\n")
print(int(time[2])$, ":", int(time[1])$, ":", int(time[0])$)
```

Result:

```
13/1/1997
20:13:21
```

See also: getFileLastModification, fillLocalTimeArray.

GetFileLastModification

Files

Aptilis 1

GetFileLastModification(FileName)

GetFileLastModification returns the date at which the file you specified was last written to.
The value returned is the number of seconds elapsed since January 1, 1970.
Such functions as fillTimeArray and fillLocalTimeArray allow you to transform this value into more explicit values for days, months and years.

Return Value:
The date of the last modification if everything was OK.
–1 in case of error and the _errno variable contains more details about the error.

Example:

```
f = "c:\\windows\\tjmail.exe" $
t = GetFilelastmodification(f$)
fillLocalTimeArray(time[], t)
print(int(time[3])$, "/", int(time[4]+1)$, "/", int(time[5])$, "\n")
print(int(time[2])$, ":", int(time[1])$, ":", int(time[0])$)
```

Result:

```
9/12/1996
11:13:26
```

See also: getFileDate, fillLocalTimeArray.

GetFileList

Files

Aptilis 1

GetFileList(DestinationArrayName[][, Filter])

getFileList will fill the array you specified with all the file names of the current directory.
The optional filter allows to get a list of files restricted to, for example, files with a given extension.
The rules that apply to the filter are the same as the ones that apply to match and this an aptilis implementation, not a system one, so that the behaviour of the filter functions is identical across the various platforms Aptilis runs on.
You can check which directory you're in with getCurrentDirectory and change to another directory with changeDirectory.
The array you specified is cleared of any previous values before being filled with the file names.

Return Value:
The number of files found and the file names in the specified array.

Example 1:

```
chdir("c:\\")
n = getfilelist(fl[])

for i=0 to n-1

print(fl[i]$, "\n")

end for
```

Result:

```
autoexec.bat
config.sys
command.com
```

Example 2:

```
chdir("c:\\aptilis")
// We just want aptilis files starting with a 'g'
n = getfilelist(fl[], "g*.e.txt")

for i=0 to n-1

print(fl[i]$, "\n")

end for
```

Result:

```
getvariable.e.txt
graphics.e.txt
getrecord.e.txt
getrecord2.e.txt
```

```
getrecordnear.e.txt
getrecord2near.e.txt
getdirlist.e.txt
```

See also: getdirectorylist

GetFileSize

Files

Aptilis 1

GetFileSize(FileName)

GetFileSize returns the size of the file you specified.

Return Value:
The size of the file if everything was OK.
–1 in case of error and the _errno variable contains more details about the error.

Example:

```
f = "c:\\windows\\win.com" $
s = getfilesize(f$)
print("Size of ", f$, " is ", int(s)$)
```

Result:

```
Size of c:\windows\win.com is 22679
```

LoadFile

Files

Aptilis 1

LoadFile(FileName)

LoadFile will simply load the whole content of the file you specified into a variable.
The path indicated should be constructed according to your platform.
Windows way: c:\texts\accounts\year96.txt
Don't forget to double the back−slashes in Windows!
Unix way: /texts/accounts/year96.txt

Neat trick: If your script resides on a server that's permently connected to the Internet (or an intranet) you can use loadFile to load web pages and FTP files too!
Make sure you check this aspect by having a look at LoadFile, in the Internet Family

LoadFile is very useful when used in conjonction with stuff to fill templates with data.

Return Value:
The whole file if everything was OK.
An empty string in case of error and the _errno variable contains more details about the error.

Example:
The file c:\templates\test.txt contains the following:

```
This is a piece of text.
That is the second line.
```

```
a = loadfile("c:\\templates\\test.txt") $
print(a$)
```

Result:

```
This is a piece of text.
That is the second line.
```

See also: saveFile.

RenameFile

Aptilis 1

RenameFile(OldFileName, NewFileName)

RenameFile renames a file.

Return Value:
0 if everything was OK.
–1 in case of error and the _errno variable contains more details about the error.

SaveFile

Files

Aptilis 1

SaveFile(FileName, Content)

SaveFile will simply create and open a file named FileName and write the content you specified into it. If an old file existed under the same name, it will be **deleted**, see AppendToFile to add things at the end of a file.
The path indicated should be constructed according to your platform.
Windows/DOS way: c:\texts\accounts\year96.txt
Unix way: /texts/accounts/year96.txt
Note the different slashes...

SaveFile can save files onto remote FTP servers using the FTP protocol. Make sure you check the internet enabled version of SaveFile in this other page!

Return Value:
Number of bytes written.
If the value is different from the length of the string you specified, then check the _errno variable for details.

Example:

```
saveFile("myfile.txt","Hello world\n-----------")
```

Result:
The file *myfile.txt* contains:

```
Hello world
-----------
```

See also loadfile, AppendToFile.

Advanced

- Lock
- Unlock

Lock

Advanced

Aptilis 1

Lock(Filename)

Lock is a crucial function if you are in a multi–user environment.
It allows you to make sure that one person, and one person only accesses a given file at any time.
The main example is a web counter where you want to make sure that all hits are taken into account. If you don't lock a file before reading and modifying it you are at risk to have a second process read an empty file insted of the modified value you intended to write. In the case of a web counter, that means that the counter can be reset to 0 unexpectedly. (And I've seen it happen, guys!)
Have a look at the counter in the examples, for a real life programing sample.
Note that locking a non existing file will result in the creation of that file, with a size of 0, if nothing is written to it. (with saveFile, apppendToFile, etc...)
Locked files are automatically unlocked at the end of your program. That means that a file cannot stay locked from one call to your aptilis program to the next, like when you call an aptilis program repeatedly at each stage of a web based data entry procedure using forms.

Lock **does not work** on network file types such as http: and ftp:. Indeed, you cannot lock a file residing on someone else's computer!

Lock is not blocking so that you can decide how long you may want to wait before you get a lock.
If you lock a file more than once (in the same programm) without unlocking it, then you need to call Unlock as many times as you've called Lock. This is in case you'd lock a file in a sub, then unlock it and expect it to be still locked after the sub has returned if you had locked it before the call.

Platform notes: Depending on your platform, the locking mechanism might work slightly differently.
Under Unix, locking doesn't prevent the locked file to be written to or read from by another process (another program that runs at the same time). But locking prevents another locking, and that's what you must do to make sure you're doing everything right. Windows is stricter and will prevent any kind of access whatsoever. So unlike under Unix, calls to LoadFile, SaveFile, etc. will fail.
Locks work on file names. You have to be careful as under Unix, the same file can have different names, through the use of links. So if a lock unexpectedly fails, make sure the file is not already locked under another name.

Subs taking advantage/affected of the locking mechanism:
Databases: AppendRecord, LoadDatabase, SaveDatabase
General File subs: AppendToFile, DeleteFile, LoadFile, RenameFile, SaveFile
SaveGifBitmap and setFont are not affected by the locking mechanism, depending on your platform. See above for details.

Return Value:
0 if everything was OK.
–1 in case of error and the _errno variable contains more details about the error.

Example:

```
path = "myFile.txt" $
```

```
t = getTime()

// Let's say we will gladly want to wait 5 seconds
failed = 0
while lock(path$) = -1
if getTime() - t > 5
failed = 1
break
end if
end while


if failed = 0
n = loadFile(path$)
saveFile(path$, n + 1)
unlock(path$)
print("n=", n, "\n")
end if
```

Result:

```
45
```

See also: unlock.

Unlock

Advanced

Aptilis 1

Unlock(Filename)

Unlock is a crucial function if you are in a multi−user environment.
It unlocks a file locked by Lock.
If you lock a file more than once (in the same programm) without unlocking it, then you need to call Unlock as many times as you've called Lock. This is in case you'd lock a file in a sub, then unlock it and expect it to be still locked after the sub has returned if you had locked it before the call. See Lock for more details, including platform idiosyncrasies and a programing example.

Flow control

- Case
- Default
- Else
- End
- If
- Return
- Select
- Sub

Case

Flow control

Aptilis 1

Case

case is the statement that you use within 'select' blocks as entry points for the different values of the control variable indicated on the 'select' line.
Negative values are allowed since Aptilis 2

For more deltails and some examples, see select.

Default

Flow control

Aptilis 1

Default

default is used in select blocks to indicate what actions to take when you have exhausted all the possibilities you want to take care of.

See select for more details and some examples.

Else

Flow control

Aptilis 1

Else

Else is the statement that can be put in the middle of an if block.
All the lines *before* the 'else' will be run if the condition on the 'if' line is true. Code *after* the else will be run if the condition is false.

See the full description and examples on the if page.

End

Flow control

Aptilis 1

End

end is used to close for, if, while, select and sub blocks.
'repeat' is closed with 'until'.

See for, if, while, select and sub.

If

Flow control

Aptilis 1

If

if expr
  line of code
  line of code
.
.
end if

Or:

if expr
  line of code
  line of code
else
  line of code (alternate)
  line of code (alternate)
end if

if allows your programme to execute one or several lines only if a condition is true.
With 'else' you can indicate some alternate lines to run when the condition is false.

Example 1:

```
a = 12
if a <20
print("a is less than 20")
end if
```

Result:

```
a is less than 20
```

Example 2:

```
a = 12
if a <20
print("a is less than 20")
else
print("a is greater than or equal to 20")
end if
```

Result:

```
a is less than 20
```

Example 3:

```
a = 21
if a <20
print("a is less than 20")
else
print("a is greater than or equal to 20")
end if
```

Result:

```
a is greater than or equal to 20
```

Notes:

- There cannot be any sub called or code after the test as is the case in other languages. The same thing is true about else lines.
- When doing tests, if you want to do them in a string context, you have to put the dollar sign ($) right at the end of the test, and only once.
  Putting several dollar signs is incorrect:
  ~~if a = b$ and c = d$~~
  This is correct:
  if a = b and c = d $
  If you want to combine tests in different contexts, you have to use brackets:
  if (a = b) and (c = d$)
  Note where the dollar is; that is inside the brackets for the second test to be done in a string context.

Return

Flow control

Aptilis 1

Return

return expr

return is used within subs, usually at the end, to return a value.
You can use a return statement within a loop if you have reached a point where you want to exit from the sub.
A return is not necessary as you may have subs that do things but do not need to return a value.
The value returned by a sub with no return statement is not predictible.
You might also use return on its own, without a value, just to exit from a sub.
A value returned at the end of the main procedure is converted to an integer and is returned to the calling process (A fact that Unix users will appreciate).

Example

```
sub main

// This is a programm to extract the square root of a number
// The number has to be typed in by the user.

n = input(20)


// There is no square root to a negative (real) number.
if n <0
return
end if

r = ComputeSquareRoot(n)
print(r, "\n")


end main


sub ComputeSquareRoot(v)

// Actually, square root is a built-in sub. :-)
sq = Sqr(v)

return sq

end ComputeSquareRoot
```

See sub.

Select

Flow control

Aptilis 1

Select

select is a very useful statement (not a sub) which will execute one block of lines out of several other blocks depending on the value of a variable.
select is a more practical way to implement such things as:

```
if a = 1
...
end if

if a = 2
...
end if

if a = 3
...
end if

if a = 4
...
end if

etc...
```

select will use a variable to decide of what action to take, but first, it will round the variable to its integer value. The corollary to that is that the different possible actions will reflect different cases attached to integer values only.
In addition the different cases cannot be variables, they have to be static values. (This will become clear with the example). A full select block comprises the 'select' and 'end select' lines with no or several cases.
'break' statements are necessary to indicate that you want to exit the select block. You can omit break statements to have two or more cases run with the same value of a variable.

Example 1:
A number speller

```
rem the user has to enter a number
n = input(2)
select n
case 1
print("one")
break

case 2
print("two")
break

case 3
print("three")
break
```

```
default
print("errrrm, I don't know!")
end select
```

Result:

```
(The user entered 2)
two

(The user entered 3)
three

(The user entered 5)
errrrm, I don't know!
```

Example 2:
A lazy number speller

```
// the user has to enter a number
n = input(2)
select n
case 1
case 2
case 3
print("one or two or three")
break

case 4
print("four or")

case 5

print("five")
break

default
print("errrrm, I don't know!")
end select
```

Result:

```
(The user entered 2)
one or two or three

(The user entered 3)
one or two or three

(The user entered 5)
five

(The user entered 4)
four or five
```

Sub

Flow control

Aptilis 1

Sub([parameters])

sub indicates the begining of a block of Aptilis lines.
In Aptilis, a programme can be divided in subs, but only one is necessary: the 'main' sub. That is the sub that is going to be run by default.
If you are running your script from the command line, you can retrieve the command line parameters by typing 'sub main(args[])' (see example 5).
A sub must be closed by *end sub* or *end the_name_of_the_sub*.
A sub can return a value with the 'return' keyword at any moment.
Returning values allows you to use subs as functions (see example 2).
You do not need the dollar sign in the 'sub' line of a sub, but you need to put the brackets '[]' to indicate that a given variable is going to be an array. (see example 3)

Example 1:
(This is actually a complete programme)

```
sub main
print("Hello World!\n")
end main
```

Result:

```
Hello World
```

Example 2:
(This is actually a complete programme)

```
sub square(param)

return param * param

end square

sub main

c = square(3)
print(c)

// we could now say 'end main', but 'end sub' is equally good.
end sub
```

Result:

```
9
```

Example 3:
(Passing an entire array)

```
sub enumerate(param[])

n = getArraySize(param[])
```

```
for i=0 to n - 1
print(param[i]$, "\n")
end for

end enumerate

sub main

a[0] = "Homer" $
a[1] = "Marge" $
a[2] = "Bart" $
a[3] = "Lisa" $
a[4] = "Maggie" $

enumerate(a[])

end sub
```

Result:

```
Homer
Marge
Bart
Lisa
Maggie
```

Example 4:
(Passing strings)

```
sub capitalize(n)

c = asc(n$)
if c >= 97 and c <= 122
c = c - 32
end if

print(chr(c)$, mid(n$, 2)$)

end capitalize


sub main

s = "hello" $
capitalize(s$)

end main
```

Result:

```
Hello
```

Example 5:
(Command line parameters)

Assuming the script is started with 'aptilis.exe test.e.txt param1 param2 param3'

```
sub main(args[])
```

```
        for i = 0 to getArraySize(args[])-1
                print(args[i]$, "\n")
        end for

end main
```

Result:

```
C:\APTILIS\APTILIS.EXE
test.e.txt
param1
param2
param3
```

As you can see, this is also the way to determine the filename of the script (args[1]$) and the location of aptilis.exe (args[0]$).

Html

- FillForm

FillForm

Html

Aptilis 2

FillForm(form)

Fillform understands HTML. It will take an HTML form and fill it's fields with the values you want.

If you do a lot of forms, especially when you want to present users with a pre-filled form then FillForm is for you.
Most common example is when you want to get some details from a user and they forgot a compulsory field. So you re-present the form with whatever they already filled in to save a bit of frustration.
FillForm makes this process incredibly easy! It is like Stuff only far more powerful as it understands HTML!

Advantages: It makes code far more legible, it allows non-coding designers to change the look and feel of a site without touching the code, you get more robust applications (in connection with my previous point...), your program is going to be faster.

Inconvenients: None. Seriously.

How it works - once you get that you don't need to read this page again.

Advancing even more

Return value:
the filled in Form in HTML format.

Quick jumps to how tags are handled:	text fields Hidden fields, same as text fields text areas radio buttons, check boxes Drop down lists without 'values', with 'values', with 'multiple' extra notes on drop down lists

Example 1, let's stay simple:
Usually you'd have your form in a separate file. (For the designer guy to change as his/her creativity demands).
But here for legibility purposes I will keep it all in.

```
// The \ are only needed because we're in source code.
//In a separate html file, they're not needed.

myForm = "<INPUT Type=\"text\" Name=\"food\">" $


// now we just initialize a variable - note the name!
// It reflects the HTML field name!
food = "pizza" $
```

```
// now the black magic! (well)
result = fillForm(myForm$) $

print(result$)
```

Result:

```
<input type="text" name="food" value="pizza">
```

Notice anything? Aptilis took your form and added:
value="pizza"

Now let's pause on this!
Here is how fillForm works:
1. You get your form in a string **variable:**
myForm = "<INPUT Type=\"text\" Name=\"food\">" $

2. You set Aptilis variables which names mirror the names of the HTML form elements to the value you want to have inserted. (Only one here in our example) As in:
food = "pizza" $

3. You call fillForm. It will return the form, filled.
result = fillForm(myForm$) $

In effect fillForm will go through your form for each form element (also called field) and see if it has any local variable **in the current sub** whose **name** is the same as the element it's looking at.
If it does then your value is going to be inserted in in proper HTML!

Cool hey? Here are some more examples for you to get more excited...

Example 2:
This time we will load the form from a file.

HTML form saved in 'form2.html':

```
<textarea name="userSays" rows="10" cols="40">this is going to be wiped out</textarea>
```

Code:

```
myForm = loadFile("form2.html") $

userSays = "Wow we're typing comments\nin here!" $

result = fillForm(myForm$) $

print(result$)
```

Result:

```
<textarea name="userSays" rows=10 cols=40>Wow we're typing comments
in here!</TEXTAREA>
```

Example 3, radio buttons:
Notice the 'checked' that has been added in the right radio button.
It is important that your radio buttons all have a unique value each within a given set.
Note: radio buttons belong to the same set if they have the same name.

HTML form saved in 'form3.html':

```
Your Favorite tastes:
<p>
<input type="radio" name="favTaste" value="sweet" /> Candy, fizzy drinks<br />
<input type="radio" name="favTaste" value="salty" /> Meat, French fries<br />
<input type="radio" name="favTaste" value="bitter" /> Guiness<br />
```

Code:

```
myForm = loadFile("form3.html") $

favTaste = "salty" $

result = fillForm(myForm$) $

print(result$)
```

Result:

```
Your Favorite tastes:
```

<input type="radio" name="favTaste" value="sweet"> Candy, fizzy drinks
 <input type="radio" name="favTaste" value="salty" checked> Meat, French fries
 <input type="radio" name="favTaste" value="bitter"> Guiness

Example 4, drop down list:
We are now going to do a simple drop−down list.
Note that we're not using any value in the 'OPTION' tag. (See next example for that).

HTML form saved in 'form4.html':

```
Your Favorite colour:
<p>
<select name="favouriteColour">
<option>orange</option>
<option>purple</option>
<option>marine</option>
</select>
```

Code:

```
myForm = loadFile("form4.html") $

favouriteColour = "marine" $

result = fillForm(myForm$) $

print(result$)
```

Result:

```
Your Favorite colour:
<P>
<select name="favouriteColour">
<option>orange</OPTION>
<option>purple</OPTION>
<option Selected>marine</OPTION>
</SELECT>
```

Example 5, drop down list again:
this time there are values in the 'OPTION' tags.

HTML form saved in 'form5.html':

```
Your Favorite colour:
<p>
<select name="favouriteColour">
<option value="1">orange</option>
<option value="2">purple</option>
<option value="3">marine</option>
</select>
```

Code:

```
myForm = loadFile("form5.html") $

// also: int(2) $
favouriteColour = "2" $

result = fillForm(myForm$) $

print(result$)
```

Result:

```
<P>
<select name="favouriteColour">
<option value="1">orange</OPTION>
<option value="2" Selected>purple</OPTION>
<option value="3">marine</OPTION>
</SELECT>
```

Example 6, drop down list last:
Sometimes a list can have several of it's options selected at the same time.
(You ususally use [Ctrl]–Click to achieve this)
FillForm understands this too, in a way that is symetric to how you retrieve multiple selections: in an array. Here we're going to select two out of the three options we have.

HTML form saved in 'form5.html':

```
Languages you know:
<p>
<select name="languages" size="3" multiple="multiple">
<option>English</option>
<option>French</option>
<option>Aptilis</option>
```

```
</select>
```

Code:

```
myForm = loadFile("form6.html") $

languages[0] = "French" $
languages[1] = "Aptilis" $
// And so on if more are needed. Do it in any order!

result = fillForm(myForm$) $

print(result$)
```

Result:

```
Languages you know:
<P>
<select name="languages" size=3 multiple>
<option>English</OPTION>
<option Selected>French</OPTION>
<option Selected>Aptilis</OPTION>
</SELECT>
```

Notes on Drop down lists: (SELECT)
– in the code that reads: <OPTION>purple</OPTION>
'purple' is called the caption.
– Sometimes an option tag has a value: <OPTION value="24">purple</OPTION>
If an option tag has no value, then the caption is also the value.
– When you assign a Drop–down value you must use the caption if there is no 'value=' thinggy in the tag.
Otherwise make sure you use the value!
– You do not need to use the </OPTION> tag. fillForm is smart enough.
– You can use a mixture of valuefull and valueless OPTIONs in the same drop down list.
– fillForm will do multiple selections, even if you do not have the 'Multiple' flag in your 'Select'. What may happen on the browser if you do that is not certain. Depending on brands and models, the browser may only take one value (the first or the last) or crash, and possibly take your whole computer with it if not your whole LAN and the world. But hey.
– In multiple selects, the order of your array has no importance.

Advancing even more
– You could also use the stuff predefined sub prior to calling fillform. That would be useful if you don't know in advance what options to put in a drop down list.

See also: stuff.

Input and Output

Input

Input and Output

Aptilis 1

Input(MaxCharactersToGet)

Input returns at most 'MaxCharactersToGet' character(s). The characters are taken from the console, that is someone has to type them!
More characters might be typed in, but only the first 'MaxCharactersToGet' of them will be retained.
The user has to press the **Return** key to end an entry.
Input cannot be used in Web programmes as all the characters available from the console (standard input for techies) have already been pumped in, in order to retrieve any form field.
Using Input in a Web script would stall it.

Return value:
A string, typed in by the user.

Example

```
a = input(20) $
print(a$)
```

Result:

```
(The user has to type something now!!)
hello!
```

Output

Aptilis 1

Output(Expr1, Expr2, Variable1, Variable2, etc...)

Output outputs expressions or variables to the standard output exactly as Print does.
The only difference is that on Windows platforms, output is not going to try to be clever and replace all occurences of "\n" by "\r\n".
This can be very annoying when you're outputing binary data such as pictures, where the data *must* be left unchanged.

This may work but is **BAD** and may not work most of the times on a Windows platform:

```
g = loadFile("image.gif") $
print("Content-type: picture/gif\n\n")

print(g$)
```

This is **better** and portable:

```
g = loadFile("image.gif") $
print("Content-type: picture/gif\n\n")

output(g$)
```

See Print for examples and the value returned.

Print

Aptilis 1

Print(Expr1, Expr2, Variable1, Variable2, etc...)

Print outputs expressions or variables to the standard output. (That's the screen when used from a console or DOS screen, or the web page returned after a form has been submited to an aptilis programme from the Web.)
Remember to add a '$' (Dollar sign) at the end of an expression to get the string value, except after string litterals (like "Hello") where print is smart enough to recognize them as strings.
You can pass one or several parameters to print, and you must separate them with commas if you pass more than one, or you can concatenate all the string parameters with '+'.
Using commas allows you to switch between string and numeric context and back if needed.
You can also pass arrays, and print will print all the values of the array.
Important: On windows platforms, print is going to try to be smart and will replace occurences of "\n" by "\r\n". If you do not wish this behaviour, use Output instead which is the same as print, except that it leaves everything unchanged and this what you want for binary data such as pictures.

Return value:
The total number of characters printed.

Example 1:

```
print("Hello World!")
```

Result:

```
Hello World!
```

Example 2:

```
a = 12
print("The value of a is: ", a)
```

Result:

```
The value of a is: 12.0000
```

Example 3:

```
a = 2
print("a = ", a, "\n")
print("The square of a = ", a * a)
```

Result:

```
a = 2.0000
The square of a = 4.0000
```

Example 4:

```
a = "Hello" $
```

```
print("Numeric value: ", a, "\n")
print("String value: ", a$)
```

Result:

```
Numeric value: 0.0000
String value: Hello
```

Example 5:

```
t[0] = "Hello" $
t[1] = "Bonjour" $
t[2] = "Guten Tag\n" $
t[3] = "Buongiorno" $
print(t[])
```

Result:

```
HelloBonjourGuten Tag
Buongiorno
```

Note that the '\n' at the end of 'Guten Tag' allows a line break.

Notes:
Some characters cannot be typed directly, because they would cause the line or the string to be broken, instead they are represented by special codes:
\n Line Feed (print to next line)
\r Carriage return, might be needed for DOS files before a '\n'
\t Tab character
\" double quotes
\\ backslash
\0 ASCII 0
\xcc, where cc stands for an hexadecimal value. This one allows you to get any character from 0 to 255. For example 'A' would be coded '\x41'. That's ASCII code 65, written as 41 in hexadecimal.

Internet

- HTTPLoad
- HTTPPostLoad
- LoadFile
- LoadPostFile
- SaveFile
- Advanced
- E-mail

HTTPLoad

Internet

Aptilis 2

HTTPLoad(url,headers[])

Although LoadFile allows you to load documents from web servers, you cannot specify the HTTP headers of your request.
HTTPLoad allows you to do so, by specifying a key based array of name/value pairs:

```
h["Accept"] = "*.*" $
h["Accept-Language"] = "en-us" $
h["User-Agent"] = "Aptilis/2.0" $
h["Host"] = _Host + ":80" $
h["Connection"] = "Close" $
h["Cache-Control"] = "no-cache" $
```

and so on...

If you want to specify some data to send and use the POST method, use HTTPPostLoad.

Specifying a Url
Here is an http:// Url: "http://www.cnn.com/"

You can use IP numbers and port numbers as well to override the default ports for http:// (80), such as:
"http://www.xnn.org:8080/news/secret.html"

In the case of web pages, the document returned contains the page itself and its header. The end of the header is after the occurrence of the header delimiter: "\n\n", or possibly "\r\n\r\n". I usually remove the "\r" 's anyway:

```
file = replace(file$, "\r", "") $
```

One of the applications of this is to be able to see the source of pages that might not be kept by your browser, and allow you to see how 'they do it'. You can also 'nick' the content of different pages and re−arrange them in a page of your own. But that's quite naughty. :−)

Return Value:
The whole file if everything was OK.
An empty string in case of error and the _errno variable contains more details about the error.

See also: HTTPPostLoad, loadPostFile, loadFile, saveFile.

HTTPPostLoad

Internet

Aptilis 2

HTTPPostLoad(url,headers[],formFields[])

Although LoadFile allows you to load documents from web servers, you cannot specify the HTTP headers of your request.
HTTPLoad allows you to do so, by specifying a key based array of name/value pairs:

```
h["Accept"] = "*.*" $
h["Accept-Language"] = "en-us" $
h["User-Agent"] = "Aptilis/2.0" $
h["Host"] = _Host + ":80" $
h["Connection"] = "Close" $
h["Cache-Control"] = "no-cache" $
```

and so on...

HTTPPostLoad will use the post method, and add the variables you specified to the request. To the web server, your request will look exactly as if it were coming from a web browser.

```
formfield["name"] = "teebo" $
formfield["creditCardNumber"] = "12345678910" $
page = httpPostLoad("http://www.someserver.com/cgi-bin/order.cgi", h[], formfield[]) $
```

Important note:
Even if you specify a content−length and a content−type in your request, they will be overwritten by Aptilis to reflect the appropriate content−type and length of your request as per the POST method.

Specifying a Url
Here is an http:// Url: "http://www.cnn.com/"

You can use IP numbers and port numbers as well to override the default ports for http:// (80), such as:
"http://www.xnn.org:8080/news/secret.html"

In the case of web pages, the document returned contains the page itself and its header. The end of the header is after the occurrence of the header delimiter: "\n\n", or possibly "\r\n\r\n". I usually remove the "\r" 's anyway:

```
file = replace(file$, "\r", "") $
```

One of the applications of this is to be able to see the source of pages that might not be kept by your browser, and allow you to see how 'they do it'. You can also 'nick' the content of different pages and re−arrange them in a page of your own. But that's quite naughty. :−)

Return Value:
The whole file if everything was OK.
An empty string in case of error and the _errno variable contains more details about the error.

See also: HTTPLoad, loadPostFile, loadFile, saveFile.

LoadFile

Internet

Aptilis 1

LoadFile(Url[, UserName[, Password[, Mode]]])

This is the special page for LoadFile, when you use URLs, instead of file names to load files. If you only load files from your local hard disk, see LoadFile

LoadFile understands the following URL types:
– file:// (normal files, check the File names topic)
– http:// to load a web page
– https:// to load a web page over SSL encryption
– ftp:// to get a file from an FTP server

Of course, to use the http:// and ftp:// types to access remote files, your computer needs to be connected permanently to the Internet or an intranet. If you are running a web server or an FTP server on your machine and want to retrieve your own files using http:// or ftp:// then of course, you do not need a connexion to a network.

If you want to use the POST method when loading an HTTP URL instead of GET (Which is limited to a couple hundred characters), use loadPostFile instead.

Specifying a Url
Here is an http:// Url: "http://www.cnn.com/"
Here is an ftp Url: "ftp://ftp.freeware–madness.org/pub/games/fry–em.zip"

You can use IP numbers and port numbers as well to override the default ports for http:// (80) and ftp:// (21), such as:
"http://www.xnn.org:8080/news/secret.html"
or:
"ftp://123.45.67.101:1040/pub/games/fry–em.zip"

In the case of web pages, the document returned contains the page itself and its header. The end of the header is after the occurrence of the header delimiter: "\n\n", or possibly "\r\n\r\n". I usually remove the "\r" 's anyway:

```
file = replace(file$, "\r", "") $
```

One of the applications of this is to be able to see the source of pages that might not be kept by your browser, and allow you to see how they do it. You can also 'nick' the content of different pages and re–arrange them in a page of your own. But that's quite naughty. :–)

When do I need to specify a username, password, and mode?
Those are needed when you access an FTP server. Indeed, FTP servers will not grant you access, unless you identify yourself. Fortunately, a lot of public servers allow you to login using the login 'anonymous' and your e–mail address as a password.
Here are the default values assigned to these parameters if you don't specify them:

UserName: "anonymous"
Password: "" (No password)
Mode: "binary"

The mode indicates how you want the file to be transfered. "Binary" indicates that you want the file to be transferred absolutely unchanged. "Ascii" asks the FTP server to try and be smart about how to store/send text files. As far as I am concerned I always use binary, and then use Replace to get the proper line separators.
That's "\n" under Unix, and "\r\n" under Windows. I start by removing all the "\r" and then replace all the "\n" by "\r\n" if I am aiming at a PC target.

LoadFile is very useful when used in conjonction with stuff to fill templates with data.

Return Value:
The whole file if everything was OK.
An empty string in case of error and the _errno variable contains more details about the error.

Example:

```
a = loadfile("http://www.mw-interactive.co.uk/") $
print(a$)
```

Result: (Note the header)

```
HTTP/1.1 200 OK
Date: Tue, 18 Aug 1998 12:47:44 GMT
Server: Apache/1.2.4
Last-Modified: Tue, 17 Mar 1998 15:55:52 GMT
ETag: "c2a0e-225-350e9d08"
Content-Length: 549
Accept-Ranges: bytes
Connection: close
Content-Type: text/html

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Mediaworks Interactive</title>
</head>

<body bgcolor="#FFFFFF">

<img src="mwi.gif" />

<hr />

Welcome on our server.<br />
```

Etc...

See also: loadPostFile, saveFile, HTTPLoad, HTTPPostLoad.

LoadPostFile

Internet

Aptilis 1

LoadPostFile(Url[,Vars])

LoadPostFile extends the capabilities of loadFile, to load the result of a CGI programm, using POST and not GET as loadFile does.
POST allows more parameters to be passed. Indeed, GET is often limited to two hundred characters on some servers, and not much more on others.
LoadPostFile can pass variables in one of two ways:
– It will send all the variables available in the sub it's been called from.
OR:
– It will pass all the variables from a key based array passed as an optional parameter. Use this method if there are some variables in your sub that you do not wish to pass to the server.

Return Value:
A string containing the response from the server, in general a web page, including the header.
An empty string in case of error and the _errno variable contains more details about the error.

Example: 1
Using the variables of the calling sub

```
sub main

// This examples uses example n.3
// Retrieving field data

// All those settings are valid on my test PC

url = "http://127.0.0.1/cgi-bin/aptilis.exe" $
file = "c:\\webshare\\scripts\\simple.e.txt" $
usersaid = "Bonjour le monde!" $

reply = loadPostFile(url$) $

print("reply 1:\n", reply$)


end main
```

Result:

```
reply 1:
HTTP/1.0 200 Ok
Content-Type: text/html
Server: Xitami
Content-Length: 67

<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<b>you said:</b> Bonjour le monde!<body>
```

```
</html>
```

Example: 2

Using the variables from a key based array

```
sub main

// This examples uses example n.3
// Retrieving field data

// All those settings are valid on my test PC

url = "http://127.0.0.1/cgi-bin/aptilis.exe" $

vars["file"] = "c:\\webshare\\scripts\\simple.e.txt" $
vars["usersaid"] = "Bonjour le monde!" $

reply = loadPostFile(url$, vars[]) $

print("reply 2:\n", reply$)


end main
```

Result:

```
reply 2:
HTTP/1.0 200 Ok
Content-Type: text/html
Server: Xitami
Content-Length: 67

<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<b>you said:</b> Bonjour le monde!<body>
</html>
```

See also: loadFile, saveFile, HTTPLoad, HTTPPostLoad.

SaveFile

Internet

Aptilis 1

SaveFile(Url, Content[, UserName[, Password[, Mode]]])

This is the page that describes how to use SaveFile with URLs.
Savefile understands two types of Urls:
– file:// (normal files, check the File names topic)
– ftp:// to get a file from an FTP server

Apart from that SaveFile works on normal file names, but it's decribed in another page. SaveFile will simply create and open a file named FileName and write the content you specified into it. If an old file existed under the same name, it will be **deleted**.

Specifying a Url
Here is an ftp Url: "ftp://ftp.freeware−madness.org/pub/games/fry−em.zip"

You can use IP numbers and port numbers as well to override the default port for ftp:// (21), such as:
"ftp://ftp.freeware−madness.org:1040/pub/games/fry−em.zip" or:
"ftp://123.45.67.101:1040/pub/games/fry−em.zip"

When do I need to specify a username, password, and mode?
Those are needed when you access an FTP server. Indeed, FTP servers will not grant you access, unless you identify yourself. Fortunately, a lot of public servers allow you to login using the login 'anonymous' and your e−mail address as a password.
Here are the default values assigned to these parameters if you don't specify them:
UserName: "anonymous"
Password: "" (No password)
Mode: "binary"

The mode indicates how you want the file to be transfered. "Binary" indicates that you want the file to be transferred absolutely unchanged. "Ascii" asks the FTP server to try and be smart about how to store/send text files. As far as I am concerned I always use binary, and then use Replace to get the proper line separators.
That's "\n" under Unix, and "\r\n" under Windows. I start by removing all the "\r" and then replace all the "\n" by "\r\n" if I am aiming at a PC target.

Return Value:
Number of bytes written.
If the value is different from the length of the string you specified, then check the _errno variable for details.

Example:

```
saveFile("ftp://myserver.com/home/file.txt", "Hello world\n-----------", "my_unsername", "my_password", "binary")
```

Result:
The file *myfile.txt* which contains:

```
Hello world
-----------
```

has been sent to the FTP server.

See also loadfile (internet).

Advanced

Call

Advanced

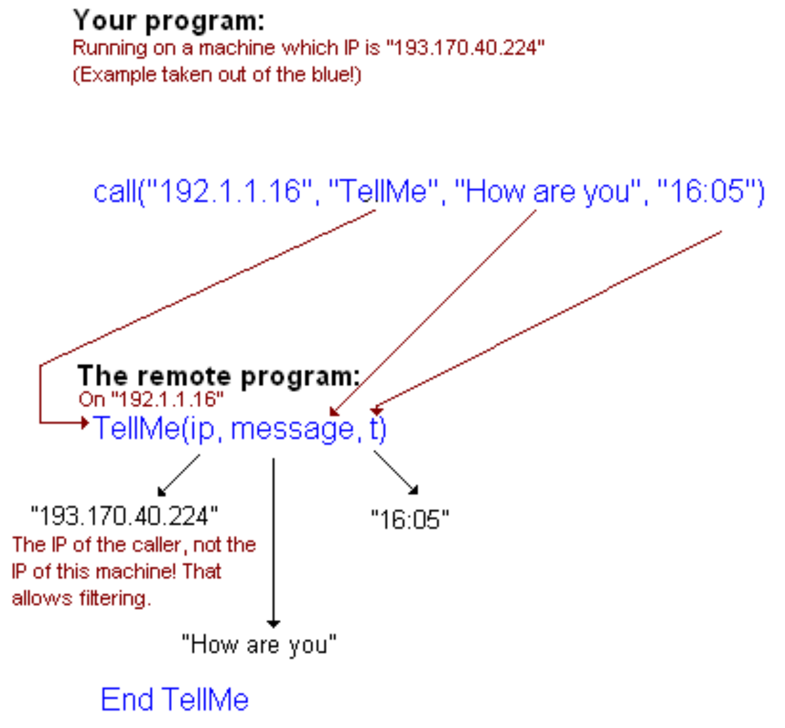
Aptilis 1

Call(IP[:port], SubName [, paramList])

Call allows you to call a sub in another aptilis program that may be running on the same machine or even on a different machine on your Intranet, or on the Internet.

The first parameter is the IP number of the machine you want to talk to. It may be a fully qulaified domain such as:
machine.domain.co.cc, for example aptilis−newx.cnn.com (imaginary!)
You can also specify a port as in:
aptilis−newx.cnn.com:63.

The second parameter is the sub name, ie. the sub you want to call (if it's in a program you haven't written, you have to have been given its name and the parameters it takes). You can have up to 32 parameters in call so that leaves you 30 to pass over.
You are limited to 30 parameters (which should be quite enough in most cases, actually) but since you can pass arrays, there is virtually no limit to the amount of information that can be passed.



Return value:
The value returned by the called sub or −1 in case of error, in which case the _errno variable will contain more details.

Example:
This is a complete example of a 'Talk' program written in aptilis.
This program is a variation of another program that I used to demonstrate the same thing, but here I added the 'input' in the 'RDspl()' sub. That makes the program, a one−way, one to one

communication device, a bit like the CB radio, where two people talk in turn.
If you remove the input in the 'RDspl()' sub, you will then need to have two windows open, one to listen, the other one to talk.
To use this program, use a DOS box (NT/95-98) or a shell and run the eqxTalk.e.txt program. Select 'L' for listening. The person you want to talk with needs to do the same thing, but will have to select 'T' instead, and then enter your IP number. He/She will have to initiate the dialog. You will then reply and so on.

```
sub main


print("Which end? (L)isten or (T)alk?\n")
m = upper(trim(input(4)$)$)$

if m = "L" $
TakeCalls(1)
print("Program stopped\n")
end if

if m = "T" $
Chat()
end if

print("Thank You. Ba-bye\n")

end sub


sub Chat()

print("Who are you talking to?")
ip = trim(input(80) $) $


repeat
m = input(4000) $

r = call(ip$, "RDspl", m$) $
if r = -1
print("errno = ", _errno$, "\n")
else
print(r$, "\n")
end if
m = trim(lower(m$)$)$
until m = "bye" $



end Chat



sub RDspl(ip, message)

print("[", ip$, "]: ", message$, "\n")
reply = input(80) $

if trim(message$) = "kill" $
```

```
takeCalls(0)
print("Now Leaving...\n")
end if

return reply$

end RDspl
```

For more details, see the Remote Sub Invocation topic.

GetLocalIP

Advanced

Aptilis 1

GetLocalIP()

GetLocalIP returns the IP of the machine your aptilis program is running on.
This can be used in conjunction with Call or TakeCalls for exemple to check that the calls are coming from the same machine if you don't want anyone else to access your script.

Return value:
Your local ip number, in a string or −1 in case of error and _errno may contain more details.

Example:

```
ip = getLocalIp()
print(ip$)
```

Result:

```
192.1.1.16
```

TakeCalls

Advanced

Aptilis 1

TakeCalls(Enabler[, port])

TakeCalls with it's first parameter set a non−zero value puts your aptilis program in listening mode. In this mode, other aptilis programs can use all the subs in your program thru the predefined sub 'Call()'. The programs calling yours may be on the same machine or on different ones across your network or across the Internet if you're connected to it.
From the moment it gets to 'TakeCalls() Your program can do nothing but reply to incoming calls. You can stop taking calls by invoking takeCalls() with a parameter of 0. But this can only be done in a sub, and in taking call mode, subs can only be run if they're called by a remote program. (Or called by a sub that has been called by a remote program, and so on).
In other words, only a call by different program may stop your program from taking calls.
The second optional parameter is the port you want your aptilis program to listen to. Usually this is set to 1108 by default, so you don't have to worry about it, but should you wish to change the default, it's possible. You may want to do so if you need several 'Listening' programs. Indeed, there can be only one program listening on any given port.

Return value:
0 or −1 in case of error, in which case the _errno variable will contain more details. Check Call for an example and the the Remote Sub Invocation topic for more details.

E−mail

GetSMTPServer

E−mail

Aptilis 1

GetSMTPServer()

implemented: aptilis v1.041 build 0015

GetSMTPServer allows you to get the current SMTP server used on your server. That may be the server indicated in the file aptilis.mail or a server you previously specified with setSMTPserver

The format for aptilis.mail is described with the sendmail commande.

Return value:
Always 0.

Example:

```
r = getSMTPserver() $
print(r$)
```

Result:

```
smtp.a_valid_isp.com
```

See also:
− SetSMTPServer to set the SMTP server. (and more details about SMTP servers)
− sendmail to send an e−mail.

ReadEmails

E−mail

Aptilis 2

ReadEmails(destinationArray[], POP_Server, UserName, Password[[[, PleaseSave], LeaveOnServer], GetHeadersOnly])

ReadEmails uses the POP3 protocol to retrieve e−mails from a mailserver.
It will retrieve all the e−mails available, store them in an array (destinationArray[]) and erase them from the server.
So you'd better save them before ending the Aptilis program.
The next three parameter are optional:
PleaseSave: If specified and different from 0 will cause ReadEmails to do a backup of the e−mails fetched on your hard drive in the current directory (named 1.mail, 2.mail etc.), but be warned, that successive calls to ReadEmails may overwrite those backups.
LeaveOnServer: If different from 0, e−mails will not be deleted from the server. Please note: If you execute ReadEmails again, you will get the same messages! *(since build 045d)*
GetHeadersOnly: If different from 0, only the e−mail headers will be returned. If larger than 1, then the first n−1 lines of the e−mail will be returned also. *(since build 045d)*

Return value:
0 if everything went well or −1 in case of error, in which case _errno may contain additional details about the error.

Example:

```
userName = "you@some_address.com" $
password = "I_am_not_telling_you" $
pop3server = "pop3.some.server.com" $

print("Now starting to read e-mails\n")

// get all mails, make no backups and delete them from the server
r = readEmails(messages[], pop3server$, userName$, password$)

print("r=", r$, "\n")
print("_errno=", _errno$, "\n")

n = getArraySize(messages[])
for i=0 to n - 1
        // store the mails in subfolder 'mails'
        saveFile("mails\\" + int(i + 1) + "-mail.txt" $, messages[i]$)
end for
```

See also: sendMail, setSMTPSever.

SendMIMEMessage

E−mail

Aptilis 2

SendMIMEMessage(MimeType, To, From, Subject, MIMEAttachmentBundle[, ReplyTo])

SendMIMEMessage − allows to specify a mime type together with an encoding, when sending an e−mail.
This will allow you to send attachments.
To create a MIME bundle of base 64 encoded files, follow the MIME specification.
A quick way to create a MIME bundle is to send a message to yourself with the text and the attachments and retrieve it with readEmails.

Mime version is 1.0

Return value:
as for sendMail.

Example:

```
attach = loadFile("aptilis-attached.txt") $
mimeType = "multipart/mixed; boundary=\"-----X-PROV-952220281157\"" $
sendMimeMessage(mimeType$, to$, from$, subject$, attach$, replyTo$)
```

See also: setSMTPSever, sendMail, readEMails.

SendMail

E−mail

Aptilis 1

SendMail(To, From, Subject, Message[, ReplyTo])

This function allows you to send e−mail messages.
In order for it to work, all you need is to put a little text file in the default directory aptilis runs in. Usually, that's the directory where you've installed aptilis. However in some cases, for instance when using aptilis for web scripts, the web server might put you in another directory. To find out in which directory you are, just use a simple script such as:

```
sub main

print("Content−type: text/plain\n\n")
print("the default directory is: ", GetCurrentDirectory() $)

end sub
```

and then call your script, saved as 'defdir.e.txt' from your web browser:
http://your.server.whatever/cgi−bin/aptilis.exe?file=C:\aptilis_programs\defdir.e.txt
Of course, you will need to adapt the URL above to your specific set−up.

There are two ways to configure the way aptilis sends e−mails:

<table>
<tr><td>Doing it once and for all in the file aptilis.mail which is going to be read the first time you use sendMail. But if you are on a foreign server, you will not be able to edit that file. The SMTP server used by default by your provider might not allow you to send e−mails if it does not know the 'from' address you gave the sendmail command.</td><td>By setting the SMTP server programmatically with the setSMTPserver command.</td></tr>
<tr><td>The text file, which must be called aptilis.mail is composed of one line that looks like this:
SMTPSever,domain
For example, if your e−mail address is joe@provider.com your aptilis.mail should look something like:
mail.provider.com,provider.com
Note that the SMTPServer part is the most important one.
An IP number is also acceptable.
If in doubt, call your internet service provider as they should know!</td><td>See the setSMTPserver command. By specifying a mail server that knows you (or the address you use as 'from'), everything should be working perfectly.
Unfortunately some ISPs (Internet Service Providers) still won't allow you to use their SMTP server if you're not connecting from one of their dial−up accounts... The solution to that is to use one of the free e−mail providers out there who, when they offer SMTP services (So that you can use Eudora for example) are less fussy about where you're coming from.</td></tr>
<tr><td colspan="2">Here is how you can find out the smtp server you use:
(Let's assume you are using Eudora 3.0, if you're not, the procedure should be similar anyway)
Click 'Tools'
Select 'Options...'</td></tr>
</table>

Select 'Hosts'
The SMTP server you're connecting to should be in there.

Of course, your computer needs to be connected to the Internet (or an intranet) if you want to be able to use this feature. If you're using a dial−up connection, make sure you connect before using this feature.

If you still can't use the required 'from' address, the optional ReplyTo parameter can help you go round that. Check the notes at the bottom of this page for more details.

Return value:
0 if everything was OK, or a string begining with a non−zero number that explains what went wrong.

Example:

```
to = "someone@some-company.com" $
from = "teebo@mw-interactive.co.uk" $
r = sendmail(to$, from$, "Aptilis mail", "Hi,\nhow are you?") $
if r = 0
print("The message has been sent\n")
else
// error
print(r$)
end if
```

Note: You can use the '\n' escape sequence to indicate a new line.

Mail messages whose 'from' address is not known to the smtp server you use may be rejected with a 553 error.
To make sure your message will be sent, use an address known to your smtp server in the 'From' parameter.
From Build 040−0009, there is an extra parameter you can put in sendmail. This fifth parameter is the 'reply−to' field, so that, even if the 'from' is incorrect, when you hit the 'reply' button in your mail program, the address used to reply to will be the one coming from that 5th parameter.
For example, if you have a web form with a from address from the person who has filled the form, use the sendmail command like this:

sendmail("you@your.isp.com", "knownAdress@your.isp.com", " a subject", "bla−bla", from$)
Say the guy's e−mail is "bob@jello.com", and that's what from$ contains. (It's coming from the web form)
Note that 'knownAddress' could be 'you'.
You will receive a message from knownAdress@your.isp.com
but when you hit 'reply', the message will be set go back to: bob@jello.com instead

See also: sendMIMEMessage, setSMTPSever, readEMails.

SetSMTPServer

E−mail

Aptilis 1

SetSMTPServer(SMTPserver)

implemented: aptilis v1.041 build 0015

SetSMTPServer allows you to override the default SMTP server indicated in the file:
aptilis.mail

The format for aptilis.mail is described with the sendmail command.

Return value:
0 if everything was OK, −1 if there wasn't enough memory to allocate your server name.

Example: (works from my home where I use Pipex as my ISP)

```
sub main

print("1. Default gateway: ", getSMTPServer()$, "\n")

setSmtpServer("smtp.dial.pipex.com")
print("2. New gateway: ", getSMTPServer()$, "\n")

r = sendMail("teebo@mw-interactive.co.uk","adw59@dial.pipex.com","test1","message du texte 1...\nEn esperant que ca marche!!!", "someone@glaine.net")
if r != 0
print("r=", r$, "\n")
print("errno=", _errno$, "\n")
end if

end main
```

SMTP is the Internet protocol used to send e−mails.
To send an e−mail, you need to connect to a server that uses the SMTP protocol. That's what most e−mail programs do, as well as the aptilis sendmail command.

This command can be particularly useful if you are on a foreign server and the SMTP server it uses won't accept the from address you specify as the 'from' parameter of the sendmail command.

By specifying a mail server that knows you (or the address you use as 'from'), everything should be hunky−dory.
Unfortunately some ISPs (Internet Service Providers) still won't allow you to use their SMTP server if you're not connecting from one of their dial−up accounts... **The solution to that** is to use one of the free e−mail providers out there who, when they offer SMTP services (So that you can use Eudora for example) are less fussy about where you're coming from.

<u>See also:</u>
− GetSMTPServer to get the current SMTP server.
− sendmail to send an e−mail. − Here is how you can find out the smtp server you use:
(Let's assume you are using Eudora 3.0, if you're not, the procedure should be similar anyway)

Click 'Tools'
Select 'Options...'
Select 'Hosts'
The SMTP server you're connecting to should be in there.

<u>Notes:</u>
To my knowledge, here are the free a−mail providers which will allow you to use their SMTP servers:
http://www.HotPop.com
The only **catch** being that they may add some text at the end of your e−mails for advertising purposes. (That's how they make their money)
If you know a service and would like to see it been mentioned here, or if the conditions of the services mentioned here have changed please don't hesistate to send me an e−mail at teebo@mw−interactive.co.uk

See also: sendMail, readEMails.

Loops

Break

Loops

Aptilis 1

Break

break is used within loops to (abruptly) exit from the loop, or indicate the end of a case block, within a select block.

break is hence used in for, while, repeat loops and select blocks.

Continue

Aptilis 1

Continue

Continue is used within loops to skip an iteration.
It can be used in for, while, repeat loops.

For

Loops

Aptilis 1

For

for *var* = *startvalue* to *lastvalue*
  line of code
  line of code
end for

OR:

for *var* = *startvalue* to *lastvalue* step *increment*
  line of code
  line of code
end for

For allows you to repeat a block of lines for a given number of times.
The for line is evaluated the first time it is entcountered so the block of lines might not be run if the condition is not true.
Example: for i=2 to 1
The *startvalue* and *lastvalue* are computed once and for all at the begining of the loop. Those values may be expressions with variables. Even if those variables are altered within the loop, that will not change the behaviour of the loop which has been defined by the *for* line.
For uses your control variable *var* and increments it by one, at the end of the *for* loop. An optional level of control is possible: instead of incrementing by one, the *step* statement allows you to choose the increment. (See example 2)
Two other statements can be used in all loops:
– break; to exit from a loop (example 3)
– continue; to interrupt the current iteration and start over from the begining of the loop, after having carried out the incrementation. (example 4)
Loops of any kind can be nested, but not intertwined.

Correct:

```
for i=1 to 5
repeat
..
..
until a = "" $
end for
```

Incorrect:

```
for i=1 to 5
repeat
..
..
end for
until a = "" $
```

Example 1:
(Don't forget the space between end and for!)

```
// Table of 4
for i=1 to 4
print(int(i)$, " * 4=", i * 4, "\n")
end for
```

Result:

```
1 * 4=4.0000
2 * 4=8.0000
3 * 4=12.0000
4 * 4=16.0000
```

Example 2:

```
for j=3 to 1 step -1
print(j, "\n")
end for
```

Result:

```
3
2
1
```

Example 3:

```
for k=1 to 10
print(k, "\n")
if k = 3
break
end if
end for
```

Result:

```
1
2
3
```

Example 4:

```
for l=1 to 4
if l = 2
continue
end if
print(l, "\n")
end for
```

Result:

```
1
3
4
```

Notes:
The test is carried out from the first iteration, the incrementation (or decrementation) is done at the end of each iteration.

Repeat

Loops

Aptilis 1

Repeat

repeat
  line of code
  line of code
until expr

repeat allows you to repeat a block of lines until a condition is fullfilled, in other words until the expression after 'until' is not zero.
All repeat loops are run **at least once.**
Two other statements can be used inside all loops:
– break; to exit from a loop (example 2)
– continue; to interrupt the current iteration and start over from the begining of the loop. (example 3)
Loops of any kind can be nested, but not intertwined.

Correct:

```
for i=1 to 5
repeat
..
..
until a = "" $
end for
```

Incorrect:

```
for i=1 to 5
repeat
..
..
end for
until a = "" $
```

Example 1:
(A classic mistake is to forget to increment the control variable, something that is done automatically by the for loop)

```
rem Table of 4
i = 1
repeat
print(int(i)$, "* 4 = ", i * 4, "\n")
i = i + 1
until i = 5
```

Result:

```
1 * 4 = 4.0000
2 * 4 = 8.0000
3 * 4 = 12.0000
```

```
4 * 4 = 16.0000
```

Example 2:

```
k = 1
repeat
print(k, "\n")
if k = 3
break
end if
k = k + 1
until k = 10
```

Result:

```
1
2
3
```

Example 3:

```
l = 0
repeat
l = l + 1
if l = 2
continue
end if
print(l, "\n")

until l > 4
```

Result:

```
1
3
4
5
```

Until

Aptilis 1

Until

Until is used to finish a repeat loop.
You have to put an expression, ie. some Aptilis code after 'until'.
If this expression, once calculated is not 0, then the loop ends and the lines after 'until' will be run.

Example:
until i = 4

See repeat for more details and some examples.

While

Loops

Aptilis 1

While

while expr
  line of code
  line of code
end while

while allows you to repeat a block of lines as long as a condition is true.
As the test is carried out from the start, a while block might never be run.
Two other statements can be used in all loops:
– break; to exit from a loop (example 2)
– continue; to interrupt the current iteration and start over from the begining of the loop. (example 3)
Loops of any kind can be nested, but not intertwined.

Correct:

```
for i=1 to 5
repeat
..
..
until a = "" $
end for
```

Incorrect:

```
for i=1 to 5
repeat
..
..
end for
until a = "" $
```

Example 1:
(A classic mistake is to forget to increment the control variable, something that is done automatically by the for loop)

```
// Table of 4
i = 1
while i < 5
print(int(i), "* 4 = ", i * 4,"\n")
i = i + 1
end while
```

Result:

```
1 * 4 = 4.0000
2 * 4 = 8.0000
3 * 4 = 12.0000
```

```
4 * 4 = 16.0000
```

Example 2:

```
k = 1
while k < 10
print(k, "\n")
if k = 3
break
end if
k = k + 1
end while
```

Result:

```
1
2
3
```

Example 3:

```
l = 0
while l < 5
l = l + 1
if l = 2
continue
end if
print(l, "\n")

end while
```

Result:

```
1
3
4
5
```

Math Functions

- Abs
- Atan
- Cos
- Exp
- Int
- Ln
- Sin
- Sqr
- Tan

Abs

Math Functions

Aptilis 1

Abs(x)

Abs returns the absolute value of a number.

Return value:
|x|, the absolute value of x.

Example 1:

```
n = abs(4)
print(n)
```

Result:

```
4.0000
```

Example 2:

```
n = abs(-3)
print(n)
```

Result:

```
3.0000
```

Atan

Math Functions

Aptilis 1

Atan(x)

Atan returns a value in radians from a tangential.

Return value:
atan(x)

Example:

```
r = atan(0.01745506492822)
print(r)
```

Result:

```
1.0000
```

Cos

Aptilis 1

Cos(x)

Cos returns the cosine of a value.
It expects to be fed angles in radians.

Return value:
cos(x)

Example:

```
r = cos(1)
print(r)
```

Result:

```
0.9998476951564
```

Exp

Aptilis 1

Exp(x)

Exp returns the exponential of a value.

Return value:
e^x (That's e to the power of x).

Example:

```
e = exp(1)
print(e)
```

Result:

```
2.718282
```

Int

Aptilis 1

Int(x)

Int returns the integer part of a value.
Note that Int truncates the decimal part of a value, which is slightly different from rounding. If you want to round a value, you should add 0.5 to it if it's a positive value or substract 0.5 from it if it's negative before using Int. For example, taking the integer part of 3.1 and 3.9 gives 3 in both cases whereas 4 would have been much more accurate in the second case.
You must then treat the integer number as a string, because when Aptilis deals with numbers, they are automatically stored as floating point values and a string of zeroes might not be what you want, especially if the value needs to be displayed.

Return value:
A string representing the integer part of x.

Example:

```
i = int(3.52) $
// The dollar is to avoid the '.0000'
print(i$)
```

Result:

```
3
```

Ln

Aptilis 1

Ln(x)

ln returns the natural logarithm of a value.
ln is not defined for negative values and 0.

Return value:
ln(x)

Example:

```
v = ln(2.718282)
print(v)
```

Result:

```
1.0000
```

Sin

Aptilis 1

Sin(x)

Sin returns the sine of a value.
It expects to be fed angles in radians.

Return value:
sin(x)

Example:

```
r = sin(1)
print(r)
```

Result:

```
0.01745240643728
```

Sqr

Aptilis 1

Sqr(x)

Sqr returns the square root of the value it has been fed with.
The square value of a number is the value which, if multiplied by itself gives the original number back. Negative numbers don't have a square value. (At least as long as you keep clear of so called imaginery numbers, which Aptilis does not implement directly.)

Return value:
the square root of x.

Example:

```
r = sqr(2)
print(r)
```

Result:

```
1.4142135624
```

Tan

Aptilis 1

Tan(x)

Tan returns the tangential of a value.
It expects to be fed angles in radians.
Tan is not defined for x = pi/2.
Tan(x) is equivalent to sin(x) / cos(x)

Return value:
tan(x)

Example:

```
r = tan(1)
print(r)
```

Result:

```
0.01745506492822
```

Miscellaneous

- Import
- Random
- Sleep
- Template
- GetProcessId

Import

Aptilis 2.4

Import dir.subdir.subsubdir.aptilisfile [as synonym]

Import is not a sub, but a directive like Sub or Template.
Import allows you to include other Aptilis scripts into your script.

For more information and some examples, see the Import topic.

Random

Miscellaneous

Aptilis 1

Random(Range)

Random returns a random number between 0 (included) and the 'range' parameter (excluded).
The random generator is initialized with a nearly random seed each time Aptilis is launched, but you can force a reinitialization, (that is using the time as the seed for new random numbers) by indicating a negative range. The absolute value of the range is then used to generate a new random number.
Random can be very useful in the implementation of games.
Warning: The Unix implementation of random might cause some bias with large ranges.

Return Value:
An integer random number n, with 0 <= n < range.

Example:

```
for i=1 to 5
print(random(10), "\n")
end for
```

Result:

```
7.0000
2.0000
2.0000
5.0000
4.0000
```

<u>Notes</u>
Random generators usually use a 'seed' or a starting point number to generate pseudo random numbers. The quality of the 'randomness' might vary from platform to platform, depending on how it has been implemented into the different Operating Systems.
No computer generated random numbers are absolutely random, and the more you run a randomizing generator, the more you are likely to entcounter a pattern, ie. a recurring suite of numbers.
By repeatedly resetting the generator, you may introduce a random factor, for example the time it takes a user to answer a question and that will lessen the risk of a pattern occuring.

Sleep

Miscellaneous

Aptilis 1

Sleep(TimeToSleepInSeconds)

Sleep suspends execution of a programme for the number of seconds indicated. No processor cycles should be used by your application while it's dormant, increasing the level of resources available to other programmes.
It returns the time in seconds as GetTime does.

Return value:
Time after completion, in seconds since Jan 1st, 1970.

Example

```
print(gettime(), "\n")
print(sleep(4), "\n")
```

Result:

```
854210692.000000
854210696.000000
```

Template

Miscellaneous

Aptilis 2.4

Template Template_Name

Template allows you to access text−blocks in your script which are outside your program code (the subs).
These text−blocks may contain HTML, XML, CSV−data (e.g. Aptilis databases) or any other plain text data. You could even use base64−encoded data for email−attachments.
Important: Your template ends with the line:
end tamplate_name
or
end template
So if you have one of these lines within your template, the parser will stop here and your template will be too short. Check the GetTemplate examples to see how to avoid this problem.

Example:

```
#!/usr/bin/aptilis.exetemplate my_templateHi, I'm a templateend my_templatesub main(args[])     print(getTemplate("my_template")$)end main
```

Result:

```
Hi, I'm a template
```

See also GetTemplate

GetProcessId

Miscellaneous

Aptilis 2.4

GetProcessId()

returns the process id of the current Aptilis instance. A possible appliance is the creation of unique temporary files.

Return value:
An integer value with the process id.

Example:

```
pid = getProcessId()
print(pid$)
```

Result:

```
-785659
```

Streams

Close

Aptilis 2

Close(Stream)

Use this command to close a stream. And make sure you do!

See Open for a primer on Aptilis streams and examples.

Return Value:
0 or −1 in case of error, in which case _errno$ will give you more details.

See also: Open, Read, ReadLine, Write, SetPosition, GetPosition
and:
LoadFile and SaveFile.

GetPosition

Streams

Aptilis 2

GetPosition(Stream)

This command will tell you what position you're at in a stream.

See Open for a primer on Aptilis streams and examples.

Return Value:
the current position in the stream specified or −1 in case of error in which case _errno$ will give you more details.

See also: Open, Close, Read, ReadLine, Write or SetPosition.

Open

Streams

Aptilis 2

Open([protocol://]Path/Domain[:port][,mode])

This predefined sub opens a stream.

What is a stream?
A stream is an **advanced** facility for you to read and write files a specified amount of bytes at a time, as opposed to LoadFile and SaveFile which read and write WHOLE files in one go without giving you a choice.

Protocols – Aptilis streams support three protocols:
– file://
– socket://
– ssl://

Ssl:// is the same as socket, except that this is using SSL encryption. As of this writing Aptilis does not offer any functionality to deal with certificates.

Modes – for file:// only
One of "read", "write", "writeread" or "readwrite" – it's NOT case–sensitive.

The same Aptilis predefined subs can be used indifferently to read and write to either files or sockets. Sockets allow computers to talk to each other over Networks.

File example: file:///c:/test/testFile.txt (PC) or file:///test/testfile (Unix)
Socket example: socket://www.cnn.com:80
As far as sockets are concerned, no assumption is made about the protocol. Because protocols usually define a port, it's up to you to explicitly specify a port (here 80, standard http port).

Yes, sockets will allow you plenty of hacking possibilities!

Each consecutive read or write operation will augment a hidden 'cursor' or position. To go back or move forward use SetPosition and GetPosition if you need to find out where in the stream you are.

Important Notes

– File streams need a mode. If you don't specify one, 'read' is assumed by default.
The 'mode' is ignored for sockets, even if you specify one.

– Files are always opened in 'append' mode. If you want to write a file from scratch, make sure you call DeleteFile on it first.

– Opening more than one stream to the same file may yield unpredictable results, especially if the file has been locked with Lock.

– To get a stream to a **locked** file, proceed as follows:
+ Lock the file – even if it does not exist yet.
+ Use open on the file name to get a stream.
+ Do your read / write stuff on the stream.
+ Once finished, call Unlock on the filename. **DO NOT** close it! Unlock will have closed it for you. Closing a file before unlocking will cause your file not to be written properly.

– On PC platforms file streams are opened in 'Binary' mode, that means you get to write exactly what you specified and read exactly what's there.

– Make sure you close streams once you're done with them. Aptilis will close opened streams for you if you've forgotten to do so, but if you don't close streams and then re–open them, you may get funny results. (For example data you've written may be lost.)

– All sockets are of TCP/IP ones and of type SOCK_STREAM. (Technical)

Return Value:
A Stream (in effect a number) to use with the other stream subs or –1 in case of error, in which case _errno$ will give you more details.

Example 1:
Files

```
sm1 = open("file://test.txt", "write")
write(sm1, "Little Red Hood")
close(sm1)

// Now we have a file called 'test.txt' in the current
// directory of our disk.


// if not protocol is specified, file:// is assumed by default
sm2 = open("test.txt", "read")
setPosition(sm2, 7)
piece = read(sm2, 3) $

print("I read: ", piece$, "\n")

close(sm2)
```

Result:

```
I read: Red
```

Example 2:
Opening a socket and playing with the http protocol. Here we will get 500 characters from the CNN website. Use LoadFile if you want the entire web page.

```
s = open("socket://www.cnn.com:80")

write(s, "GET / HTTP/1.0\r\nAccept: *\r\n\r\n")
p = read(s, 500) $
print(p$)
```

```
close(s)
```

Result:

```
HTTP/1.0 200 OK
Date: Thu, 13 Dec 2001 01:56:30 GMT
Server: Netscape-Enterprise/4.1
Last-Modified: Thu, 13 Dec 2001 01:56:31 GMT
Expires: Thu, 13 Dec 2001 01:57:31 GMT
Cache-Control: private, max-age=60
Content-Type: text/html
Age: 72
Via: HTTP/1.0 ntl_site (Traffic-Server/3.5.7-10686 [uScMsSf pSeN:t cCMi p sS])

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang=
```

See also: Close, Read, ReadLine, Write, SetPosition, GetPosition
and:
LoadFile and SaveFile.

Read

Streams

Aptilis 2

Read(Stream, LengthToRead)

This stream command allows you to read data (in the form of strings) from a stream.
See readLine to read lines.

See Open for a primer on Aptilis streams and examples.

Return Value:
A string of as many characters as you specified, unless there wasn't enough data, in which case you string will be shorter than the requested length.
An empty value returned to you may indicate an error, that _errno$ will give you more details about.

See also: Open, Close, ReadLine, Write, SetPosition, GetPosition
and:
LoadFile and SaveFile.

ReadLine

Streams

Aptilis 2

ReadLine(Stream)

This stream command allows you to a get line of text from a stream, from the current position. Aptilis will gather all the characters from the specified stream until it finds a line separator (LineFeed) '\n', ie. a character which ASCII code is 10.
The '\n' character will not be removed and you will get it at the end of your line. In files generated by a PC (as opposed to a Unix system) each '\n' character may be preceded by a '\r' or ASCII character 13, also called Carriage Return'. This too will have been left at at the end of your line, before the '\n'. To get rid of those, you may use the trim command.

See Open for a primer on Aptilis streams and examples.

Return Value:
A line of text, (which may only contain "\n" or "\r\n" if it's a blank line) and an empty string "" at the end of the file.

Example 1: *We're using a file called 'rltest.txt' that contains:*

```
Archer
Kirk
Picard
Cisko
Janeway

h = open("rltest.txt", "read")

repeat
l = readLine(h) $
if len(l$) = 0
break
end if
print("{{{{", l$, "}}}}\n")
until 0

close(h)
```

Result: (note the }}}} on the following line, because the '\n' has been preserved.)

```
{{{{Archer
}}}}
{{{{Kirk
}}}}
{{{{Picard
}}}}
{{{{Cisko
}}}}
{{{{Janeway
}}}}
```

Example 2: *This time instead of a file, we're going to be using a socket. However sockets usually connect you to another program on another machine. This program will most likely follow a protocol. In other words we have to indicate what we want, that's what the write is for here. Port 80 is the usual http port.*

```
h = open("socket://www.cnn.com:80")

// http speak to indicate we want the default page at root level.
write(h, "Get / HTTP/1.0\r\n\r\n")

repeat

l = readLine(h) $
if len(l$) = 0
break
end if
print("{{{{", l$, "}}}}\n")
until 0


close(h)
```

Result: (note the }}}} on the following line, because the '\n' has been preserved.)

```
{{{{HTTP/1.0 200 OK
}}}}
{{{{Date: Tue, 29 Jan 2002 21:41:08 GMT
}}}}
{{{{Last-Modified: Tue, 29 Jan 2002 21:41:09 GMT
}}}}
{{{{Expires: Tue, 29 Jan 2002 21:42:09 GMT
}}}}
{{{{Cache-Control: private, max-age=60
}}}}
{{{{Content-Type: text/html
}}}}
{{{{
}}}}
{{{{<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
}}}}
{{{{<html lang="en">
}}}}
{{{{<head>


..... ok I skipped a bit here for legibility .......

{{{{</body>
}}}}
{{{{</html>
}}}}
```

See also: Open, Close, Write, Write, SetPosition, GetPosition
and:
LoadFile and SaveFile.

SetPosition

Streams

Aptilis 2

SetPosition(Stream, Position)

Use this command to position yourself in a stream so as to read only the required characters or write to the position of your choice.

See Open for a primer on Aptilis streams and examples.

Return Value:
the previous position or −1 in case of error in which case _errno$ will give you more details.

See also: Open, Close, Read, ReadLine, Write or GetPosition.

Write

Streams

Aptilis 2

Write(Stream, StringToWrite)

This stream command allows you to write data (in the form of a string you specify) to a stream.

See Open for a primer on Aptilis streams and examples.

Return Value:
The nuber of bytes written or −1 in case of error in which case _errno$ will give you more details.

See also: Open, Close, Read, ReadLine, SetPosition, GetPosition
and:
LoadFile and SaveFile.

Strings

Asc

Strings

Aptilis 1

Asc(String)

Asc returns the ASCII code of the first character of the string you specified.

Return Value:
An ASCII code.

Example:

```
asciiCode = asc("A")
print(asciiCode)
```

Result:

```
65
```

Notes:
ASCII stands for American Standard for Characters Interchange and Interface.
Numbers from 0 to 9 are coded with values from 48 to 57 respectively, and capital letters go from 65 (A) to 90 (Z). Note that adding 32 to a capital letter ASCII code brings the lower case equivalent.
ASCII is usually standard across platforms (PC, Mac, Unix...) except for the codes after 128. This is a problem for HTML pages where an accentuated letter will look fine on the original platform but will translate to another character on an other computer. That is why it is recommended to use character entities for all accentuated characters, i.e. é for 'é' when doing HTML pages.

Cat

Strings

Aptilis 1

Cat(String1, String2, String3...)

Cat is a cheap alternative to string concatenation with '+' as in:
c = a + b $
Cat uses less memory and is faster than '+', especially when concatenating more than 2 strings together.
Programatically, however use whatever solution is more legible to you.

Return Value:
A concatenated strings made up of up to 32 parameters passed to Cat().

Example:

```
s = "Leelou" $
c = cat("The cat's name is: ", s$) $
print(c$, "\n")

p1 = "The cat " $
p2 = "thinks " $
p3 = "she is " $
p4 = "a Bengali tiger!" $

phrase = cat(p1$, p2$, p3$, p4$)$
print(phrase$, "\n")
```

Result:

```
The cat's name is: Leelou
The cat thinks she is a Bengali tiger!
```

Chr

Strings

Aptilis 1

Chr(Expr1, Expr2, Variable1, Variable2, etc...)

Chr returns one character from the ASCII code it has been given. Chr, not unlike print can take several parameters, and even arrays, which it interprets as numeric values.

Return value:
A string composed of one or more characters, from the ASCII code(s) it has been given.

Example 1:

```
print(chr(90)$)
```

Result:

```
Z
```

Example 2:

```
clearArray(ac[])
ac[0] = 72
ac[1] = 69
ac[2] = 76
ac[3] = 76
ac[4] = 79
print(chr(ac[])$)
```

Result:

```
Hello
```

Notes:
ASCII stands for American Standard for Characters Interchange and Interface.
Numbers from 0 to 9 are coded with values from 48 to 57 respectively, and capital letters go from 65 (A) to 90 (Z). Note that adding 32 to a capital letter ASCII code brings the lower case equivalent.
ASCII is usually standard across platforms (PC, Mac, Unix...) except for the codes after 128. This is a problem for HTML pages where an accentuated letter will look fine on the original platform but will translate to another character on an other computer. That is why it is recommended to use character entities for all accentuated characters, i.e. é for 'é'.

Format

Strings

Aptilis 1

Format(PadCharacter, mBefore, nAfter, value)

Format, as its name implies, will format a number into a string containing at least nBefore characters on the left of the decimal point and exactly After characters after the decimal point.
You can specify a padding character to be used on the left of the decimal point, you will usually want a space or zero.
– If the pad string is empty then nFormat will be likely to fail in putting at least nBefore characters before the decimal dot.
– If the number is so great that it cannot fit in nBefore characters, it will **NOT** be truncated, and you will not get what you expected.
– If the pad string contains more than one character, only the first one will be used.

Return Value:
A formatted string containing a number.

Example:

```
print(Format("0", 4, 2, 0), "\n")
print(Format("0", 3, 1, 1.5), "\n")
print(Format("x", 3, 1, -1.5), "\n")
print(Format("0", 1, 1, -1.5), "\n")
print(Format(" ", 2, 1, 1.59), "\n")
print(Format(" ", 2, 1, -1.49), "\n")
print(Format(" ", 1, 3, 1.49), "\n")
print(Format("0", 3, 0, 1.5), "\n")
print(Format(" ", 2, 2, 1998), "\n")
```

Result:

```
0000.00
001.5
-x1.5
-1.5
1.6
-1.5
1.490
001
1998.00
```

GetCharAt

Strings

Aptilis 1

GetCharAt(String, Pos)

GetCharAt is used to get the ASCII code of a character inside a string without resorting to a costly getSubString.

The ASCII code of the character in position Pos from string String.

Example:

```
s = "Aptilis" $
for i = 1 to len(s$)
print(int(i)$, " ", getCharAt(s$, i), "\n")
end for
```

Result:

```
1 65.000000
2 112.000000
3 116.000000
4 105.000000
5 108.000000
6 105.000000
7 115.000000
```

See also Len.

GetSubString

Strings

Aptilis 2

GetSubString(String, FromWhere[, HowManyCharacters])

GetSubString extracts a sub−part of a string (substring). If no length is given, then the substring goes from *FromWhere* to the end of the mother string.
If *FromWhere* is negative, it is set to 1, and *HowManyCharacters* is decreased by the absolute value of the original *HowManyCharacters*.
if *HowManyCharacters* is negative, characters are taken from *FromWhere*, then to the left.

Return Value:
A substring of the specified string.

Example 1:

```
a = GetSubString("short bread", 7) $
print(a$)
```

Result:

```
bread
```

Example 2:

```
t = "One two three" $
a = GetSubString(t$, 5, 3) $
print(a$)
```

Result:

```
two
```

Notes:
For left, right and GetSubString, indexes start at one.
if the start position goes beyond the length of the expression given, an empty string is returned.
If the length given goes beyond the expression given, only the available characters are returned, and **no** padding occurs.

GetTemplate

Aptilis 2.4

GetTemplate (Template_Name)

Returns a template defined with the template directive somewhere in your Aptilis script.

Return Value:
The template in a string or an empty string in case of error (_errno$ may tell you why).

Example 1:

```
#!/usr/bin/aptilis.exetemplate PageTop<html><head><title>My Homepage</title></head><body>end PageToptemplate PageBottom<hr size="1" width="80%">&copy; 2004 by Donald McRonald</body></html>end PageBottomsub main      pageTop    = getTemplate("PageTop") $   pageBottom = getTemplate("PageBottom") $        print("Context-type: text/html\n\n")    print(pageTop$, someContent$, pageBottom$)end main
```

Example 2:

```
#!/usr/bin/aptilis.exetemplate Database"Mr.","Bond","James","MI6""Mr.","Hunt","Ethan","IMF""Mr.","Mulder,"Fox","FBI"end templatesub main        n = parseDatabase(db[], getTemplate("Database")$)       print("Good Morning, ", getField(db[1]$,0)$, " ", getField(db[1]$,1)$)end main
```

Result:

```
Good Morning, Mr. Hunt
```

Example 3:

```
#!/usr/bin/aptilis.exetemplate TestTemplatethis won't terminate the template: end TestTemplatethis won't, too:end TestTemplate Testbut this terminates it:end templatethis would terminate it, too - if it wasn't already terminated by the previous "end template":end TestTemplatesub main    print(getTemplate("TestTemplate")$)end main
```

Result:

```
this won't terminate the template: end TestTemplatethis won't, too:end TestTemplate Testbut this terminates it:
```

See also Template

Instr

Strings

Aptilis 1

Instr([FromWhere,] String1, String2)

Instr looks for a string into another string. It returns the position of the searched string if it has been found, otherwise it returns **zero**.
When Aptilis is dealing with strings, indexes start at 1. For example, the index of "tw" in "two" is 1, and the index of "wo" in "two" is 2.
The optional index specified before the two strings allows you to search from a given position. That is especially useful when you try to find several occurences of the same substring in one given string.

Return Value:
The index of the substring being searched or 0 (zero) if it has not been found.

Example 1:

```
e = "can of beans" $
print(instr(e$, "of"))
```

Result:

```
5.0000
```

Example 2:

```
e = "can of beans of Canada" $
print(instr(7, e$, "of"))
```

Result:

```
14.0000
```

Notes:
Even if a start position is given, the index returned will be a count from the first character.

See also:
Rinstr

Join

Strings

Aptilis 1

Join(ArrayName[], Connection)

Join puts all the elements of an array together in a string. It uses a linking element which can be an empty string.

Return value:
The new string.

Example:

```
sub main

a[0] = "zero" $
a[1] = "one" $
a[2] = "two" $
a[3] = "three" $
a[4] = "four" $
a[5] = "five" $

print(join(a[], "")$, "\n")
print(join(a[], ", ")$, "\n")
print("A unix record:\n", join(a[], ":")$, "\n")

end main
```

Result:

```
zeroonetwothreefourfive
zero, one, two, three, four, five
A unix record:
zero:one:two:three:four:five
```

<u>See also:</u> Separate, which does the opposite of Join.

Left

Strings

Aptilis 1

Left(String, HowManyCharacters)

Left returns the specified count of characters of the string given, starting from the left.
If count goes beyond the length of the string given, the return value will be no longer than the original string. (There is no padding to meet the length requested).

Return Value:
A substring of the string given, which length will be at most, the one specified.

Example:

```
a = "Aptilis" $
b = left(a$, 3) $
print("3 letters of Aptilis: ", b$, "\n")
```

Result:

```
3 letters of Aptilis: Apt
```

Len

Strings

Aptilis 1

Len(String)

Len returns the length of a string.

Return Value:
The length of the string specified.

Example:

```
a = "Hello" $
c = len(a$)
print("Length of ", a$, " = ", c)
```

Result:

```
Length of Hello = 5.0000
```

Lower

Strings

Aptilis 1

Lower(String)

Lower returns the expression it has been given with all letters set to lower case.

Return Value:
The same string, with letters A–Z set to lower case.

Example:

```
print(lower("AbCdEfG")$)
```

Result:

```
abcdefg
```

Notes:
Depending on the platforms, lower might return incorrect results for accentuated characters.

Match

Strings

Aptilis 1

Match(String, Pattern)

Match returns a non zero value if the pattern matches the string.
The point in using 'match' as opposed to a straightforward equility test, is that the pattern can contain wildcards, like '*' tp specify any group of characters and '?' to specify any one character.
That allows you to match strings that are not necessarily exactly similar.

Return Value:
1 if the pattern matches the string, 0 (zero) otherwise.

Example 1: *Word ending in 'e'?*

```
print(match("hello", "*e"), "\n")
print(match("bye", "*e"), "\n")
```

Result:

```
0.0000
1.0000
```

Example 2: *Word starting with an 'a'?*

```
print(match("banana", "a*"), "\n")
print(match("apple", "a*"), "\n")
```

Result:

```
0.0000
1.0000
```

Example 3: *Word containing a 't'?*

```
print(match("clue", "*t*"), "\n")
print(match("ate", "*t*"), "\n")
```

Result:

```
0.0000
1.0000
```

Example 4: *Word containing a 't', then an 'n'?*

```
print(match("plate", "*t*a*"), "\n")
print(match("resting", "*t*n*"), "\n")
```

Result:

```
0.0000
1.0000
```

Example 5: *Word ending in 'ree', with only one letter before 'ree'?*

```
print(match("real", "?ree"), "\n")
print(match("tree", "?ree"), "\n")
```

Result:

```
0.0000
1.0000
```

<u>Notes</u>
The patterns used by 'Match' are exactly the same as the ones used by GetRecordIndexByNearKey.

Mid

Strings

Aptilis 1

Mid(String, FromWhere[, HowManyCharacters])

Mid is another name for GetSubString.

Replace

Strings

Aptilis 1

Replace(String, Target, Replacement)

Replace as its name indicates, replaces occurrences of a target substring with a given replacement.

Return value:
A new string where all the occurences of the target have been replaced.

Example:

```
a = "John went to buy a hamburger, because he likes ham." $
b = replace(a$, "ham", "cheese") $
print(b$)
```

Result:

```
John went to buy a cheeseburger, because he likes cheese.
```

Note: (especially to C programmers)
The replacement is not done directly in the string given. Instead, a new string is created.
So to replace correctly do:
var = replace(var$, target$, replacement$) $
And not: ~~replace(var$, target$, replacement$)~~

Right

Strings

Aptilis 1

Right(String, HowManyCharacters)

Right returns the last *HowManyCharacters* characters from the specified string.
If count goes beyond the length of the string given, the return value will be no longer than the original string. (There is no padding to meet the length requested).

Return Value:
A substring of the string given, which length will be at most, the one specified.

Example:

```
a = "Aptilis" $
b = right(a$, 3) $
print("3 last letters of Aptilis: ", b$, "\n")
```

Result:

```
3 last letters of Aptilis: lis
```

Rinstr

Strings

Aptilis 1

Rinstr([FromWhere,] String1, String2)

Rinstr works like Instr except that it looks for string2 in string1 starting from the right.
It returns the position of the searched string if it has been found, otherwise it returns **zero**.
When Aptilis is dealing with strings, indexes start at 1. For example, the index of "tw" in "two" is 1, and the index of "wo" in "two" is 2.
The optional index specified before the two strings allows you to search from a given position. That is especially useful when you try to find several occurences of the same substring in one given string.

Return Value:
The index of the substring being searched or 0 (zero) if it has not been found. Note that the index is relative to the begining of the string (as for Instr) not the end.

Example:

```
e = "can of beans of Canada" $
print(rinstr(e$, "of"))
```

Result:

```
14.0000
```

Notes:
Even if a start position is given, the index returned will be a count from the first character.

See also:
Instr

Separate

Strings

Aptilis 1

Separate(DestinationArrayName[], String, Separator)

Separate separates a string into its constituents separated by a separator.

Return value:
The number of element(s) found.

Example:

```
c = "one++two++three++four++five++six++seven" $
n = separate(e[], c$, "++")
for i=0 to n-1
print(e[i] $, "\n")
end for
```

Result:

```
one
two
three
four
five
six
seven
```

See also: Join, which does the opposite of Separate.

String

Aptilis 1

String(Count, Pattern)

String creates a string by repeating a pattern for the number of times specified.

Return Value:
A string composed of the repetition of a substring.

Example:

```
p = "bla" $
print(string(3, p$)$)
```

Result:

```
blablabla
```

Stuff

Strings

Aptilis 1

Stuff(String)

stuff uses a special syntax: !(varname) to insert variables in a string. You can prepare a string and indicate where you want the variable to be inserted. Stuff will replace all the occurrences of !(varName) with the value of varName. Of course you can use different variables as well as the same variable several times.
Stuff is especially useful for CGI scripts, where you may want to change the looks of an HTML page coming out of a script, without having to go back into a programme.

Return value:
A new string where the variables have been replaced by their values.

Example:
(Note that one of the variables, age, has not been defined)

```
template = "Welcome !(name). You are !(age) years old." $
name = "John" $
print(stuff(template$)$)
```

Result:

```
Welcome John. You are  years old.
```

Notes:

- The replacement is not done directly in the template given. Instead, an new string is created and returned.
  This allows you to use the template several times, in a loop for eaxmple.
- stuff is especially useful for web applications when used in conjunction with database functions. It saves the repeated printing of all the HTML so that you can concentrate on displaying the information.
  See loadfile, loaddatabase
- If you take the template from a file (with loadfile) that will allow you to separate the 'looks' from the actual programme and changing the lay−out of your web pages will only require changing the templates, not diving into your code.

Trim

Strings

Aptilis 1

Trim(String)

Trim removes blank spaces from the begining and the end of a string you pass to it. In effect, it removes characters which ASCII codes are less than 33. (Spaces, tabs, line−feeds, etc.) *à la Java*.

Return Value:
The string given minus blank characters at the end and begining.

Example:

```
test = "  Vaiko " $
print("1. [", trim(test$)$, "]\n")
print("2. [", trim("  le chien")$, "]\n")
print("3. [", trim("leelou  ")$, "]\n")
print("4. [", trim("le chat")$, "]\n")
print("5. [", trim("")$, "]\n")
print("6. [", trim("16 06 70")$, "]\n")
```

Result:

```
1. [Vaiko]
2. [le chien]
3. [leelou]
4. [le chat]
5. []
6. [16 06 70]
```

Upper

Strings

Aptilis 1

Upper(String)

Upper returns the expression it has been given with all letters set to upper case.

Return Value:
The same string with letters a−z set to upper case (capitals).

Example:

```
print(upper("Hello!")$)
```

Result:

```
HELLO!
```

Notes:

- Depending on the platforms, upper might return incorrect results for accentuated characters.
- On the Internet, writing words all in upper case is associated with shouting and can be considered rude if used to often!

Time

DoTime

Time

Aptilis 1

DoTime(ArrayName[])

Dotime is the reverse of FillLocalTimeArray. You must first fill the array with values for year, month, day, hour, minutes, and seconds and then pass the array to doTime. doTime will then return a number of seconds elapsed since January 1st, 1970.
The values to fill are:
0: Seconds (0..59)
1: Minutes (0..59)
2: Hours (0..23, 24 hour clock = military time)
3: day in month (1..31)
4: month (0..11)
5: year (full year, ex: 1998)

No other values should be present. You can use clearArray to empty an array.

Return value:
The time, in seconds.

Example:

```
day[0] = "Sunday" $
day[1] = "Monday" $
day[2] = "Tuesday" $
day[3] = "Wednesday" $
day[4] = "Thursday" $
day[5] = "Friday" $
day[6] = "Saturday" $

t = gettime()
print("time: ", t, "\n")

FillTimeArray(time[], t)

print("Heure: ", int(time[2])$, ":", int(time[1])$, ":", int(time[0])$, "\n")
print("Date: ", int(time[3])$, ":", int(time[4]+1)$, ":", int(time[5])$, "\n")

print("\nDay: ", day[time[7]]$, ", year day:", int(time[6])$, "\n")
print("The reverse is: ", DoTime(time[]), "\n")
```

Result:

```
time: 853352573.000000
Heure: 18:22:53
Date: 15:1:1997

Day: Wednesday, year day:15
The reverse is: 853352573.000000
```

See also: getTime, fillTimeArray, fillLocalTimeArray.

FillLocalTimeArray

Time

Aptilis 1

FillLocalTimeArray(DestinationArrayName[], TimeValue)

FillLocalTimeArray fills the specified array with meaningful local time values calculated from a standard time value.
The array given to the sub is emptied of any previous values that it might have contained.
Standard time values express the time as the number of seconds elapsed since January 1st, 1970.
Such values are returned by getTime, getFileDate and getFileLastModification.
The time returned by this function is the current time in your country (or the country your system thinks it's in) to get GMT time, use fillTimeArray.
Here are the locations of the values returned:
0: Seconds (0..59)
1: Minutes (0..59)
2: Hours (0..23) 24 hour clock=military time
3: day in month (1..31)
4: month (0..11)
5: year (full year, ex: 1997)
6: day in year (1..366)
7: day in week (0..6, Sunday is 0)

Return value:
The time given to the function in seconds and meaningful values in the array specified.

Example:

```
t = gettime()
fillLocalTimeArray(time[], t)
print("The time is (In the UK) ")
print("(As of this writing!):\n")
print(int(time[3])$, "/", int(time[4]+1)$, "/", int(time[5])$, "\n")
print(int(time[2])$, ":", int(time[1])$, ":", int(time[0])$)
```

Result:

```
The time is (In the UK) (As of this writing!):
13/1/1997
21:12:11
```

FillTimeArray

Time

Aptilis 1

FillTimeArray(DestinationArrayName[],TimeValue)

FillTimeArray fills the specified array with meaningful GMT time values calculated from a standard time value.
The array given to the sub is emptied of any previous values that it might have contained.
Standard time values express the time as the number of seconds elapsed since January 1st, 1970.
Such values are returned by getTime, getFileDate and getFileLastModification.
The time returned by this function is the standard time or 'Greenwich time' also know as 'GMT'. To get the time in your country (or the country your system thinks it's in) use fillLocalTimeArray.
Here are the locations of the values returned:
0: Seconds (0..59)
1: Minutes (0..59)
2: Hours (0..23) 24 hour clock=military time
3: day in month (1..31)
4: month (0..11)
5: year (full year, ex: 1997)
6: day in year (1..366)
7: day in week (0..6, Sunday is 0)

Return value:
The time given to the function in seconds and meaningful GMT values in the array specified.

Example:

```
t = gettime()
fillTimeArray(time[], t)
print("The GMT time is:")
print("(As of this writing)\n")
print(int(time[3])$, "/", int(time[4]+1)$, "/", int(time[5])$, "\n")
print(int(time[2])$, ":", int(time[1])$, ":", int(time[0])$)
```

Result:

```
The GMT time is:(As of this writing)
13/1/1997
21:12:11
```

GetTime

Time

Aptilis 1

GetTime()

GetTime returns the time, in effect the number of seconds elapsed since January 1st, 1970.
Two other functions, ie. fillTimeArray and fillLocalTimeArray can transform this kind of value into something more meaningful: hours, minutes, seconds, days, months, years...

Return value:
The time, in seconds.

Example:

```
t = gettime()
print("Number of seconds since 1/1/70\n")
print("(As of this writing)\n")
print(t)
```

Result:

```
Number of seconds since 1/1/70
(As of this writing)
853188770.0000
```

Variables

Var

Variables

Aptilis 2

Var variable_name

This allows you to 'declare' the variables at the beginning of a sub.
Although Aptilis does not require variables to be declared, if you choose to do so with 'var' then any undeclared variable found by the parser will trigger an error. This can help find bugs such as 'nTOtal' typed incorrectly in places instead of 'nTotal'.

Example:

```
var tax
tax = "16 %" $
print("Tax: ", tax$)
```

GetGlobalVariablesList

Aptilis 2.1

GetGlobalVariablesList()

GetGlobalVariablesList returns an array with the names of all global variables.
The variable names will even appear in the list if they were not initialised yet and if they are defined in imported files.

Return value:
A string array with the variable names.

Example:

```
sub Init()

        _homepage = "http://www.aptilis.com/" $

end Init

sub main()

        Init()
        globalvars[] = getGlobalVariablesList()
        for i = 0 to getArraySize(globalvars[])-1
                print(globalvars[i]$, "\n")
        end for

end main
```

Result:

```
homepage
errno
ENV
```

See also GetLocalVariablesList

GetLocalVariablesList

Variables

Aptilis 2.1

GetLocalVariablesList()

GetLocalVariablesList returns an array with the names of all variables that are currently in scope.

Return value:
A string array with the names of all local variables.

Example:

```
sub main(args[])

        localvars[] = getLocalVariablesList()
        for i = 0 to getArraySize(localvars[])-1
                print(localvars[i]$, "\n")
        end for

end main
```

Result:

```
args
localvars
i
```

See also GetGlobalVariablesList

GetVariable

Variables

Aptilis 1

GetVariable(String)

Getvariable retrieves the value of a variable. As it takes an expression as a parameter, variables which names are not known in advance can be retrieved effortlessly.

Return value:
A string if the variable exists, otherwise an empty string.

Example:

```
my_var = "Hi Everybody!!" $
a = getVariable("my_" + "var") $
print("a=", a$, "\n")
```

Result:

```
a=Hi Everybody!!
```

XML

GetXMLField

XML

Aptilis 2

GetXMLField(PARSED_xml_in_a_string, path)

GetXMLField will extract a field from a piece of XML.

Note that you cannot extract directly from raw XML.
You must first parse the XML with the parseXML command.

The path is the form name1:[index].name2[:index]... etc.
See the examples to see what this mean. If indexes are not specified, then they're assumed to be 0, so the path will bring back the first element that it matches.

Indexes start at 0.

Return Value:
The field specified or an empty string if not found.

All the examples will load a file called 'test.xml' that contains the following:

```
<container technology="tj">
<n_individuals>4</n_individuals>
<individual>
<name>Teebo</name>
<favfood>Pizza</favfood>
</individual>

<individual>
<name>Mickey</name>
<favfood>Cheese</favfood>
</individual>

<individual>
<name>Donald</name>
<favfood>Bread Crumbs</favfood>
</individual>

<individual>
<name>Casimir</name>
<favfood>Gloobeboolga</favfood>
</individual>

</container>
```

Example 1:
Simple loading, parsing and field extraction

```
xml = loadFile("test.xml") $

// We always reset _errno, as loadFile may have set it.
_errno = "" $
pxml = parseXML(xml$) $
```

```
field = getXMLField(pxml$, "CONTAINER.INDIVIDUAL.NAME") $

print(field$)
```

Result:

```
Teebo
```

Example 2:
Same thing (simplified) but with error checking code.

```
xml = loadFile("test.xml") $

// We always reset _errno, as loadFile may have set it.
_errno = "" $
pxml = parseXML(xml$) $

// Here's how to test for errors
if len(xml$) = 0 and len(_errno$) > 0
// We have an error!
print(_errno$)
end if
// etc...
```

Example 3:
Same as example 1, only now we are using an index.

```
xml = loadFile("test.xml") $

// We always reset _errno, as loadFile may have set it.
_errno = "" $
pxml = parseXML(xml$) $

field = getXMLField(pxml$, "CONTAINER.INDIVIDUAL:2.NAME") $

print(field$)
```

Result:

```
Donald
```

Example 4:
A bit more useful, we now extract all the names.
Note the use of the 'stuff' command.

```
xml = loadFile("test.xml") $

// We always reset _errno, as loadFile may have set it.
_errno = "" $
pxml = parseXML(xml$) $


// First we get the number of fields:
n = getXMLField(pxml$, "CONTAINER.N_INDIVIDUALS")

// Using Stuff is the easier way to build a complex key.
key = "CONTAINER.INDIVIDUAL:!(i).NAME" $
for i=0 to n - 1
```

```
// int it so as to avoid all the 000000000

i = int(i) $
field = getXMLField(pxml$, stuff(key$)$) $

print(field$, "\n")
end for
```

Result:

```
Teebo
Mickey
Donald
Casimir
```

See also parseXML, GetXMLTagAttributes, and stuff.

GetXMLTagAttributes

XML

Aptilis 2

GetXMLTagAttributes(DestinationArray[], PARSED_xml_in_a_string, path)

GetXMLTagAttributes retrieves the attributes of a tag. As with GetXMLField you need to get some XML, parse it with ParseXML and then indicate a path to the tag you're interested in.
Warning: Only up to 32 parameters can be retrieved. Any tag containing more than that will cause an out of memory error.

The path is the form name1:[index].name2[:index]... etc.
If indexes are not specified, then they're assumed to be 0, so the path will bring back attributes for the first element that it matches.

Indexes start at 0.

Return Value:
A **Key based array** of name/value pairs for the attributes of this tag or an empty array if there were no attributes.

In the following example we will load a file called 'test.xml' that contains the following:

```
<container technology="tj">
<language quality="super" funpotential="top">Aptilis</language>
</container>
```

Example:
Simple loading, parsing and field extraction

```
xml = loadFile("test.xml") $

// We always reset _errno, as loadFile may have set it.
_errno = "" $
pxml = parseXML(xml$) $


getXMLTagAttributes(attrs[], pxml$, "CONTAINER.LANGUAGE")


for i=0 to getArraySize(attrs[]) - 1
k = getNextKey(attrs[]) $
print("(*) ", k$, ":", attrs[k$]$, "\n")
end for
```

Result:

```
(*) quality:super
(*) funpotential:top
```

See also parseXML and GetXMLField.

ParseXML

XML

Aptilis 2

ParseXML(xml_in_a_string)

ParseXML takes a string that contains some XML and parses it. That means it will transform it into another string so that getXMLField will have an easier job getting fields out of it.

The string returned, although containing the same information as the original one is of no interest to you directly. If printed for example, it would look like gobbledeegook.

Return Value:
A special string to use with getXMLField. On error, an empty string is returned (So check for it's length) and _errno$ contains more details. (Such as a parsing error and the line where it occurred.)
See also getXMLField.

Function index

Appendix

License

THE APTILIS LICENSE

http://www.aptilis.com/

Preamble

The Aptilis is derived from the BSD Licence.
Why a new Licence? Simply to cater for the specific aims the author has set for Aptilis, which are mainly to offer a simple, consistent, compact and easy to deploy solution.
Aptilis specific changes are identified with the caption: *(Aptilis special close)*

The Licence

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the Aptilis Solutions nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

* *(Aptilis special close)* Modified version of the product where modifications affect features being added or removed (for example a new command) need imperatively to be distributed under another name so as to keep the feature set consistent. Any such modified product must however bear the mention, 'based on Aptilis Technology' and carry this licence, unchanged. Modified versions where modifications aim at either increase performance or code manageability with impact on features may be redistributed under the same name.

* *(Aptilis special close)* As corollary to the previous measure, any new feature (such as a new command) will require written consent from the copyright holder for inclusion into Aptilis.

(Aptilis special close)
Aptilis also includes the work of the following groups:

Freetype: http://www.freetype.org/
Please read: www.freetype.org/patents.html

OpenSSL: http://www.openssl.org/

```
    This product includes software developed by the OpenEvidence Project
    for use in the OpenEvidence Toolkit (http://www.openevidence.org/).
    This product includes software developed by the OpenSSL Project
    for use in the OpenSSL Toolkit (http://www.openssl.org/).
    This product includes cryptographic software written by Eric Young
    (eay@cryptsoft.com). This product includes software written by Tim
    Hudson (tjh@cryptsoft.com).

Please read the licence and legal documents included in those products.
```

(End of Aptilis special close)

Credits

Aptilis includes the Freetype engine, a nice piece of work that allows you to use True Type Fonts in your Aptilis bitmaps!
www.freetype.org

The SSL−Support is based on the librarys of the OpenSSL project:
www.openssl.org

The PDF−version of this handbook has been created with the free software 'HTMLDOC' from Easy Software Products
www.easysw.com/htmldoc

The CHM−version of this handbook has been created with 'HTML Help SDK' from Microsoft
msdn.microsoft.com/(...)

The history of Aptilis

Thibault Jamme about the beginnings of Aptilis:

I started thinking of the ideal CGI language when I first wrote CGI scripts back in 1995. Designers were involved in the creative loop from early on and they found the arcane syntax of PERL unmanageable. C, my favourite, was even worse. All languages at the time required libraries to be added on in order to cope with the web, for example to acquire HTML form variables. Another nightmare was portability were a program written under Windows would not even compile under Unix, and PERL was no different, in particular when it came to sending e−mails programmatically. C was very bad with Strings, you'd either keep leaking memory (and that's extremely lethal to servers such as IIS who keep programs dormant) or you'd spend ages debugging your application.

Aptilis is based around a compiler that tranforms Aptilis source code in a string of operands and operators, piled−up together a la 'reverse polish' notation.

Aptilis addresses all those short comings by providing a similar feature set to Linux and Windows, and many things necessary for web deployment are built−in: HTML Form variables acquisition, Output of graphics with True Type support, XML, SSL, support for templates to separate business logic from graphic design and layout, etc. Aptilis makes beginners and non−programmers' lives easier.

Aptilis' first name was equinox after an album from Jean−Michel Jarre. However too many things were called Equinox, from ski companies to celtic sects. When it came to change the name I wanted to use the word 'habilis' after the first humans who used tools. Unfortunately, this was the name of a cleaning company in Belgium. 'hab' had to become 'apt'.

Changes from Aptilis 1 to Aptilis 2 (until build 044)

```
New predefined subs
-------------------

- SetArrayDimensions(Array[], dim1size, dim2size, ...)
  Up to 31 dimensions may be specified.
  returns total number of elements in array

- GetArrayDimensions - fills an array with the description of
  a multi-dimension array

- GetSubstring, but it's just another name for the already existing Mid

- GetCharAt to get the ASCII code of a character inside a string without
  resorting to a costly GetSubString

- Cat, a cheap alternative to string concatenation with '+' as in:
```

```
c = a + b $
can now also be:
c = cat(a$, b$) $
```

- ParseDatabase will parse a database from a string as opposed to from a file. (Which allows you to get your databases from other sources than files)

- SortDatabase(db[], whatField, sortType) (Example in sdb.e.txt) to sort a database using one of its fields.

- Stream subs: open, close, read, write, setPostion, getPosition. File handles (that may also be sockets in disguise) appear in Aptilis. What a shock!

- LoadPostFile extends the capabilities of loadFile, to load the result of a CGI program, using post and not get. LoadPostFile dumps either the local variables of the sub it's been called from, or the values of a key based array passed as an extra, optional parameter.

- HTTPLoad and HTTPPostLoad allow you to specify the Headers of an HTTP request.

- ReadEmails to read emails through a POP3 gateway.

- SendMimeMessage - allows to specify a mime type together with an encoding:

```
attach = loadFile("aptilis-attached.txt") $
mimeType = "multipart/mixed; boundary=\"-----X-TJTECH-952220281157\"" $
sendMimeMessage(mimeType$, to$, from$, subject$, attach$, replyTo$)
```

  The body of the message needs to be encoded accordingly.
  Mime version is 1.0

**** The new Aptilis Star Command! ****

- FillForm is a very powerful command that 'stuffs' an HTML form. It fills the value parameters of input tags and puts 'checked' and 'selected' flags where needed in checkboxes/radioboxes and SELECTs respectively! This is very useful when a form needs to be filled with values to be edited!

**** XML Support ****

- ParseXML will parse a piece of XML.

- GetXMLField will extract a field from a parsed XML string.

- GetXMLTagAttributes to get the attributes of a tag.

New features

- Multiple dimension arrays. (Note that they cannot expand dynamically because aptilis wouldn't know how - So an out of range subscript causes the programm to stop with an error.)

- main params

  The entry sub (by default: main) can now receive command line parameters:

```
  ex:  sub main(args[])

       end sub

  Note that the environment variables are available in the global array _ENV[""]
  and the number of command line parameters are availaible through:
  n = getArraySize(args[])

- Enhanced and rationalized error messages.

- Customized copyright message
  If there is a file called "aptilis-message.txt" in the directory where aptilis
  started, then this file is displayed in place of the original aptilis copyright
  message.

- Run-time error messages give out a call tree

- Select/case: case now takes negative paramaters as well

- var keyword
  This allows you to 'declare' the variables at the begining of a sub.
  Although aptilis does not require variables to be declared, if you choose
  to do so with 'var' then any undeclared variable found by the compiler
  will trigger an error. This can help find bugs such as 'nTOtal' typed
  incorrectly in places instead of 'nTotal'.

- Subs can now return arrays! Including thru the call/takeCalls mechanism!!

- HTTP 1.1 now supported! This was badly needed to retrieve web pages
  served by virtual web servers (servers that have several domains on
  the same IP number)

- Compile only flag.
  From the command line, adding an exclamation mark at the end of the aptilis
  file will signify to aptilis to check the program for errors, but not to run it.
  (You can't have aptilis program file names ending with an '!' now).

- More comment types:
  #
  '
  as well as the old //
  and... // can now be at the end of a line
  /*
     multiline comments
     now supported!
  */
  The rem comment has been removed as it was far too confusing.


- Removed features

  + From build 35, the 'entry' feature has been removed. That used to allow
  you to selecet an alternative to the 'main' sub when calling a program
  from the web. Indeed a hacker could have called any sub with the parameters
  of his / her choice.
```

Structural changes (No effect on coding, unless clearly stated otherwise)

```
- New 'for' implementation.
  The for loop run-time code has completly been re-implemented.
  It should be a bit faster and fixes a bug with aptilis 1.xxx which
  could not use a key indexed variable to loop.

- Completly new sub calling mechanism.
  This is to make the code re-entrant, in view to multithreading.
  The implementation of takeCalls is also affected.

- Call (RSI) uses the new sub calling mechanism.
  Performance should not be affected, but the new implementation
  makes the code more efficient, more legible and more maintainable.
  It also paves the way for future developments.

- LoadDatabase has been re-written. Build 27. Parsing is now halted on
  any error.
```

Improvements

```
- Plenty of memory leaks fixed on exit caused by errors.
  (Thank you very, very much Code Guard!!!)
  (Code guard is a utility that came with my Borland C++ and it's just
   out of this world!)

- 'For' speed should be better.

- Significant code re-writes. (optimization, reliability, maintenance.)

- Variable context masking done at compile time (to save a few clocks at
  run-time) (Don't worry about it - it's fairly technical.)

- Operation in numeric context sped up. (inlining, internal.)

- Operations in string context have been optimized for cases where outcome
  is false.

- Less redundancy with the (internal) socket functions.

- Less redundancy between Database file functions and pure file functions (internal).

- Replace is SO much faster and efficient.
```

Known issues

```
- a line looking like: (with boolean operators)
  r8 = a < b and 1 < 3
  will produce a syntax error. This is due to a quirk in the compiler
  which for the moment identifies assignments on a separate pass.

  Workaround: (It won't slow down your code)
  r8 = (a < b and 1 < 3)

- Unix version: GetFileLastModification returns the same value as GetFileDate

- Comments at the end of lines don't work all the time // and they should.
```

Issues solved

```
-------------

- Substractions in string context may cause crash instead of generating an error.
  Fixed as of 039.

- Separate used to return an empty array when the target was longer than
  the substrate. It now returns the whole substrate when the target
  is empty or longer than the substrate.

- Fixed missing header problems on loadFile loading URLs - this may cause
  some of your code to work differently, you may now need to explicitly remove
  headers from documents loaded from the Web, using the http protocol.
  ex: f = loadFile("http://www.cnn.com/") $

- Memory optimization at the end of database parsing.
```

Build History

```
-------------

   030 Fixed empty fields in databases that were ignored and shifted the rest
       leftward.
   031 Fixed 2 gasping huge memory over-allocation problems in parseDatabase.
   032 Added 'ReadLine' Command.
   033 Fixed 'Textarea' bug in fillForm.
   034 Used generic write functions for saveDatabase (Can now use ftp:// too.)
   035 Compiled with Freetype 1.5 (5 Nov 2002)
   036 Added file name in error message when script hasn't been found and fixed
       loading script problem on Unix. (13 Nov 2002)
   037 Added XML <TAG/> syntax. (26 Nov 2002)
   038 Fixed Freak XML parsing bug where parsed xml string thought it was bigger than
       it actually was.
       Re-factored http fetcher code - to higher standards, better performance and enhanced
       legibility.
       Added full versioning info in http headers when using built-in http fetcher.
       Fixed rare memory leak on wrong call to LoadPostFile.
       (12 Feb 2003)
   039 Fixed ClearArray bug that was WAY off the mark... An incorrect value was used
       to set the array's size.
       Fixed crash when reporting an inapproriate operator in a String context operation.
       (13 Feb 2003)
   040 Fixed rare replace bug that caused the end of the substrate not to be copied
       when the substrate was too short.
       Support for Transfer-Encoding: chunked in HTTP replies now supported. (02 Apr 2003)
   041 Support for SSL added. Fixed a bug on chunked HTTP with big documents.
       (29 Apr 2003)
   042 Fixed XML GetTagAttributes that was broken on tags near the end of the data
       block. (15 Jul 2003)
   043 Call to SSL_CTX_free, not much effect on leakage on shutdown. (2 Aug 2003)
       Note that the SSL leakage on shutdown should not be a problem.
       Also fixed problem where assigning an empty array to an array didn't clear up the
       destination array.
   044 Fixed XML parsing bug where the last field of an XML block may have been
       irretrievable.
       Also added XML comments support as well as the skipping of the Directives.
       (8 Sep 2003)
   045a Very first phase of MySQL interface integration. Adds predefined sub MySQLScramble.
       This may or may not stay inside Aptilis, or may stay but may not be documented.
```

045b Max: First Open Source build with new license information.
Libs used: OpenSSL 0.9.7c; FreeType 1.3.1.
045c Two new subs: GetLocalVariablesList and GetGlobalVariablesList.
045d Max: Fixed Bug in ReadEmails and added two optional parameters ´don't delete´ and ´headers only´.
046 Fixed stream 'write' bug.
*047*1* GetTemplate implemented, WARNING contains debugging code that outputs junk - so not to be considered release material. Big parts of the parser have been completely re-written.
*047*2* Added support for <nick@server.com> "Nicks Realname" in sendmail, so that the sender's real name should appear in the recipient's message.

Aptilis on the web

The Aptilis website:
http://www.aptilis.com

The Aptilis support section with discussion forums and many example scripts
http://www.aptilis.com/support/

Learn how You can help the Aptilis project
http://www.aptilis.com/how2help/

Aptilis at SourceForge
http://sourceforge.net/projects/aptilis

Copyright