Hecl - The Mobile Scripting Language [http://www.hecl.org/]

David N. Welton, hecl.org <hecl@googlegroups.com>

Table of Contents

Introduction

The Hecl Programming Language is a high-level scripting language implemented in Java. It is intended to be small, extensible, extremely flexible, and easy to learn and use.

Why Hecl? Hecl is intended as a complement to the Java programming language, not a replacement. It tries to do well what Java doesn't, and leaves those tasks to Java for which it is best suited, by providing an API that makes it easy to tie the two together. Hecl aims to be a very immediate language - you can pick it up and start doing useful things with it quickly - even people without formal training. Hecl is easy to learn. Where Java is verbose and rigid, Hecl is forgiving and quick to write. For instance, **`System.out.println("Hello World");`** vs **`puts "Hello World"`** - 41 keystrokes (shifted letters count double) versus 22. Hecl is built to "scale down" [http://www.welton.it/articles/

scalable_systems.html] - especially in terms of its users, meaning that it is very quick to learn, and can be quickly put to productive use even by those who are not programmers by trade.

This makes Hecl ideal for large applications written in Java that would like to provide a user friendly scripting interface, rather than, say, a clunky XML based system. Examples include: scripted web pages, command/control logic in long running applications, and, I'm sure, many environments I've never considered. Instead of a simple, static configuration file, you can give your users the power to program portions of the system to do things that you hadn't thought of when you wrote the software originally.

Hecl is a small language with a minimal core. The idea is to provide only what's necessary in the language itself, and as needed, add in extensions for specific tasks. Core Hecl is small enough to run on my Nokia 3100 cell phone as a J2ME application, presenting the interesting possibility of writing scripts, or at some point, maybe even scripting entire applications, for devices running embedded Java.

Contributions in the form of code, ideas, suggestions, or even donations are welcome. Hecl is still growing, so your thoughts are important, and you can help shape the language's future. You can download the latest source code via git [http://git-scm.com/] from github: http://github.com/davidw/hecl/tree/master.

Hecl is available under the liberal Apache 2.0 open source license. Which says, more or less, that you may use Hecl in your own applications, even if they are not open source. You have to give the authors credit, though. Read the license itself to clear up any doubts. Incidentally, I don't see the license as being incompatible with the GPL, so feel free to utilize Hecl in your GPL product (I have added a note to this effect in the NOTICE file that must accompany products using the Hecl code).

I owe thanks to a lot of people for Hecl. First and foremost the creator of the Tcl programming language, Dr. John Ousterhout. While I have attempted to improve some things that I did not care for in Tcl, it is obvious that the simple, extremely flexible command-based approach that Hecl takes is derived from Tcl. I also borrowed some ideas from the (mostly defunct) Jacl implementation of Tcl in Java. Many thanks are also due my friend Salvatore Sanfilippo, with whom I have spent many hours discussing Hecl, Tcl, and the philosophy of programming languages in general. And of course, I owe a huge debt of gratitude to my wife, Ilenia, who puts up with all the hours I spend in front of "that damn computer".

Installation / Tools & Editors

Hecl is easy to compile and install as a standard J2SE application.

Note

This is only necessary if you'd like to work on Hecl's source code. If all you want to do is use it, or install it on your mobile phone, these steps aren't necessary - the standard distribution contains everything you need.

Note

On Microsoft Windows, you must add your java `bin` directory to the system path, otherwise you will get strange, seemingly unrelated errors!

1. **Dependencies:**

   You will need to install these software packages if you want to rebuild Hecl.

   Obviously, you need a Java SDK - we suggest something recent like 1.5 or 1.6, as that's what most of the Hecl developers have.
   Depending on what you want to utilize Hecl for, you'll want one or both of the Sun Java Wireless Toolkit [http://java.sun.com/javame/downloads/index.jsp] or Android SDK [http://code.google.com/android/download.html].

Hecl uses the Apache Ant [http://ant.apache.org/] build system, so you need to install that as well to compile Hecl. If you are on Windows, this may require a bit of extra work. Here are some instructions for setting up Ant on Windows [http://blogs.sun.com/rajeshthekkadath/entry/installing_ant_on_windows].

2. To compile the standard, J2SE version of Hecl, do this:

   ```
   ant packageCommandline
   ```

3. You should now have a Hecl.jar file in the `jars/j2se/` directory. To run it, do this:

   ```
   java -jar jars/j2se/Hecl.jar
   hecl> puts "hello world"
   hello world
   ```

4. If you want to check your installation of Hecl, you can run the test suite to make sure everything checks out:

   ```
   java -jar jars/j2se/Hecl.jar tests/suite.hcl
   ```

   An (incomplete) performance test is also supplied so that you can compare numbers if you're curious, or want to hack on Hecl to improve its speed:

   ```
   java -jar jars/j2se/Hecl.jar tests/performance.hcl
   ```

If you're going to be recompiling (you don't need to recompile if you're only going to be writing scripts, though!) Hecl for Java ME, you also need to follow these instructions.

1. You'll need to download and install Sun's "Java Wireless Toolkit", here: http://java.sun.com/products/sjwtoolkit/index.jsp.

2. You also need to make Hecl aware of the WTK. For instance, in my installation, the `settings.xml` file has the following:

   ```
   <!-- General WTK settings -->
   <!-- make it fit for your local installation -->
   <property name="my.wtk.home" value="/opt/WTK2.5.2"/>
   ```

   You would need to change `/opt/WTK25.2` to wherever the WTK is located on your system.

See the section called “Hecl and Java ME” for further information on installation and use with J2ME.

Tools & Editors

Even if you're not interested in recompiling Hecl to hack on the source code, to write Hecl scripts, you need some kind of editor for programming. Hecl is close enough in syntax to the Tcl programming language that editors that support Tcl work well with Hecl.

- **Emacs**' Tcl mode is easy to use: **`M-x tcl-mode`**, or if you want to associate `.hcl` scripts with tcl-mode, add this to your `.emacs` file:

  ```
  (setq auto-mode-alist
        (cons '("\\.hcl$" . tcl-mode)
       auto-mode-alist))
  ```

- **Eclipse** is popular for working on Java projects. It also has a Tcl mode that can be utilized for Hecl scripts like so:

  Editing Hecl scripts with Eclipse

  1. Install the DLTK from the following URL: http://download.eclipse.org/technology/dltk/updates, from within Eclipse.
  2. After downloading you must associate the `*.hcl` files with the Tcl Source Editor.

     Windows # Preferences # General # Editors # File Associations
  3. Than make a new file type by adding the `*.hcl` file type, and select the Tcl Source Editor.

Hecl Tutorial

Note

This is a general tutorial on the Hecl language - if you're looking for the tutorial about Hecl on Java ME, it's here: Java ME Tutorial.

Like many people, I enjoy taking something and experimenting with it before going and reading the instructions! With this in mind, I have written a brief tutorial that you can use to start exploring Hecl on your own.

To launch the interactive Hecl shell:

```
java -jar ./jars/j2se/Hecl.jar
```

This will give you a prompt: `hecl>` where you can type in commands.

Of course, we would be out of place not to begin with the famous "Hello, World". Behold:

```
puts "Hello, World"
```

Hecl is based on the notion of commands, which take any number of arguments. The **puts** command takes one argument, a string, and prints it out.

Like all programming languages, Hecl provides variables that may be used to store information. Here, we set a variable, `rating`, and then print it out in the midst of a string. This is called "interpolation", and is a convenient way of creating new strings.

```
set rating 10
puts "Hecl, from 1 to 10: $rating"
```

Something else we notice in the above examples is that we use double quotes "" to group a series of things. In Hecl, commands and their arguments are separated by spaces. Since **puts** only takes one argument, a string, we use the quotes to group several words together in order to pass them as one string to the command. Many languages require quotes to delineate a string, but in Hecl that is not necessary if the string has no spaces in it. For instance, **`puts helloworld`** is legitimate.

Something else visible in the above command is that Hecl commands occur one per line, and that no line ending is necessary, as in languages like C where lines end in a semicolon. In Hecl, the semicolon is optional, and can be used to put more than one command on a line:

```
puts "hello" ; puts "world"
```

Another way of grouping multiple words in Hecl is with braces: {}. Hecl does not automatically perform any substitution on the variables or commands grouped within braces, as it does with quotes.

```
puts {The $dollar $signs $are printed     literally$$ - no substitution}
```

Aside from the dollar sign, which returns a reference to the value of a variable, it is also possible to utilize the results of one command as the input of a second command. For example:

```
set rating 10
puts "Rating:"
puts [set rating]
```

In this case, we pass the results of the **set** command to the **puts** command. In reality, **`set rating`** is just a long way of writing `$rating` but it's a good example.

Like everything else in Hecl, we perform math operations as commands:

```
puts "2 + 2 = [+ 2 2]"
```

In the example, the + takes two arguments, adds them together and return the result, which is then printed out by the **puts** command.

In order to choose between one or more

```
set temp 10
if { < $temp 0 } {
    puts "It's freezing"
} else {
    puts "Not freezing"
}
```

"while" loop command:

```
set i 0
while { < $i 10 } {
    puts "i is now $i"
    incr $i
}
```

Lists:

```
set foo [list a b c]
set bar {a b c}
lappend $foo d
lappend $bar d
set foo
# Returns 'a b c d'
```

```
set bar
# Returns 'a b c d'
```

Hash tables:

```
set foo [hash {a b c d}]
puts [hget $foo a]
# prints 'b'
puts [hget $foo c]
# prints 'd'
hset $foo c 2
puts [hget $foo c]
# prints '2'
puts $foo
# prints 'a b c 2' (although not necessarily in that order)
```

"foreach" loop command:

```
set lst {a b c d e f}
foreach {m n} $lst {
    puts  "It is possible to grab two variables at a time: $m $n"
}

foreach {x} $lst {
    puts  "Or one at a time: $x"
}
```

Create new commands with the "proc" command. In this example we create a command that prints out a numbered list. In Hecl, commands created within procs normally are only visible within that proc, and are cleaned up when the procedure exits. For exceptions to this rule, see the **global** and **upeval** commands.

```
set list {red blue green}
proc printvals {vals} {
    set num 1
    foreach v $vals {
 puts "$num - $v"
 incr $num
    }
}

printvals $list
```

Hecl is very flexible - in this example, we create a "do...while" loop command that works as if it were a native loop construct.

```
proc do {code while condition} {
    upeval $code
    while { upeval $condition } {
 upeval $code
    }
}

set x 100
set foo ""
do {
    append $foo $x
    incr $x
} while { < $x 10 }
set foo
# Returns 100 - because the loop is run once and only once.
```

Hecl Commands

These commands are part of the Hecl core and are always present.

Name

= != — Integer equality

Synopsis

= {*num1*} {*num2*} != {*num1*} {*num2*}

Description

Tests two numbers for equality, returning 1 if they are, 0 if they aren't equal. In the case of **!=**, returns 1 if they are not equal, 0 if they are equal.

Example

```
puts [= 1 1]
puts [= 0 1]
puts [= 00001 1]
puts [!= 1 1]
```

Produces:

```
1
0
1
0
```

Name

+ - * / — Basic math commands.

Synopsis

`+` {*number*} [*number*...] `-` {*number*} {*number*} `*` {*number*} {*number*} `/` {*number*} {*number*}

Description

The basic math commands take two arguments and carry out a numerical operation on them. In subtraction, the second argument is taken from the first. In division, the first argument is divided by the second.

Example

```
puts [+ 2 2]
puts [+ 1 2 3]
puts [- 10 1]
puts [* 6 7]
puts [/ 100 5]
```

Produces:

```
4
6
9
42
20
```

Name

abs acos acos asin asin atan atan cbrt ceil cos cosh cosh exp expm1 floor hypot log log10 log1p pow random round signum sin sinh sqrt tan tanh tanh — Floating point math commands

Synopsis

`command` {*number*} [*number*]

Description

Floating point math commands. The names are mostly self explanatory, corresponding to the methods found here: http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html

Note

Keep in mind that none of these commands are available in the MIDP1.0 version of Hecl, and that the following commands are available only in J2SE versions of Hecl:

acos atan ceil round exp floor sin cos sqrt log tan asin

The following are only available in versions of Hecl built for Java 1.5 and above.

cbrt cosh expm1 hypot log10 log1p signum sinh tanh

Name

after — Sleep / delayed script evaluation

Synopsis

`after` { *`milliseconds`* [*`script`*] | info [*`event`*] | cancel {*`event`*} [*`event`*...] | idle [*`script`*] }

Description

The **after** command is used, in the simple case, to simply halt evaluation of the script for *`milliseconds`* milliseconds, or execute a script after that number of milliseconds.

The after command can also be used to manage timer events in Hecl via the **info** and **cancel** subcommands, which, respectively, return information about existing events, and allow the cancellation of events.

Example

Pause for 1 second:

```
after 1000
```

Run *`script`* after 10 seconds, but do not pause execution of the main script:

```
after 10000 { puts "hello, later world" }
```

Cancel all timers:

```
for {set i 1} {< $i 11} {incr $i} {
    puts [after [* 10000 [random]] [list puts "Event number $i"]]
}

after 5000

foreach e [after info] {
    after cancel $e
    puts "Event $e cancelled"
}

twait forever
```

Will produce, on average, something like the following:

```
timer#1
timer#2
timer#3
timer#4
timer#5
timer#6
timer#7
timer#8
timer#9
timer#10
Event number 9
Event number 4
Event number 10
Event timer#5 cancelled
Event timer#1 cancelled
Event timer#7 cancelled
Event timer#3 cancelled
Event timer#2 cancelled
Event timer#6 cancelled
Event timer#8 cancelled
```

Name

and — Logical and

Synopsis

`and {number} [number] [number] [...]`

Description

The **and** command takes one or more arguments, and performs a binary and on them, in sequence.

Example

```
hecl> and 1 0
0
hecl> and 1 1
1
hecl> and 1
1
hecl> and 2 4
0
hecl> and 5 1
1
hecl> and 5 4
4
hecl> and 1 2 4 8
0
```

Name

append — Append text to a variable.

Synopsis

`append` {*`varreference`*} {*`string`*}

Description

The append command takes two arguments, a variable reference, and a string to append to the variable.

Example

```
set foo "bar"
append $foo "beebop"
# The foo variable now contains the string barbeebop
```

Produces:

```
barbeebop
```

Name

break — Break out of a loop.

Synopsis

```
break
```

Description

The break command breaks out of a loop. If this command is not run from within a loop - the **while** or **foreach** commands for instance, it generates an error.

Example

```
set i 0
while { true } {
    if { > $i 100 } {
        break
    }
    incr $i
}
```

In what would otherwise be an endless loop, the break command is used to exit.

Name

bgerror — Called for background task errors.

Synopsis

`append` {*`message`*}

Description

Hecl calls **bgerror** when there is a problem in a background task (created with the **after** command, for instance).

Example

```
hecl> proc bgerror {msg} { puts "Houston, we have a problem: $msg" }
hecl> after 1 { beebop }
timer#7
```

Produces:

```
Houston, we have a problem: Command beebop does not exist
```

Name

catch — Evaluates a script, catching any errors.

Synopsis

`catch` {*`script`*} [*`varname`*]

Description

The catch command evaluates a script, and returns 0 if the script evaluated successfully. If there were errors, catch returns 1. Optionally, a variable name may be passed to the command, where the results of the script evaluation will be placed. In the case of errors, the stack trace will be placed in the variable argument. If the script executes without problems, the variable will contain the result of the script execution.

Example

```
catch nosuchcommand foo
set foo
```

Produces:

```
   {ERROR {Command nosuchcommand does not exist}}
```

Name

classof — Returns the name of the internal class of a Hecl Thing.

Synopsis

`classof {variable}`

Description

The classof command returns a string containing the Java name of the class that the Hecl variable contains internally.

Example

```
classof [+ 1 2]

classof "foo"
```

Produces:

```
org.hecl.IntThing

org.hecl.StringThing
```

Name

clock — Provides time utilities

Synopsis

`clock` [{seconds} | {time} | {format *`format`*}]

Description

The **clock** command is used to return information on the current time. With the `seconds` option, the number of seconds since January 1st, 1970 are returned. With the `time` option, milliseconds since that date are returned.

With the `format` option, the **clock** command takes a millisecond value and returns a formatted date and time. For example:

Example

```
hecl> clock format [clock time]
Fri Jun 15 13:22:20 CEST 2007
```

Name

continue — Skip to next cycle of a loop

Synopsis

```
continue
```

Description

The continue command is used within the bodies of looping commands such as **if** and **while**. When **continue** is called, execution of the loop body stops and and execution moves on to the next iteration of the loop.

Example

```
set i 0
set res {}
foreach x {a b c d e} {
    incr $i
    continue
    append $res $x
}
puts $i
puts $res
```

Produces:

```
5
```

The res variable is never appended to, so printing it out produces an empty string.

Name

copy — Copy a Hecl value.

Synopsis

`cd` {*`value`*}

Description

The **copy** command makes a deep copy of a Hecl value, whereas normally, Hecl variables contain references.

Example

```
hecl> set foo 1
set bee $foo
set bop [copy $foo]
incr $foo
puts "foo is $foo"
puts "bee is $bee"
puts "bop is $bop"
```

Produces:

```
foo is 2
bee is 2
bop is 1
```

Name

double — Cast as a double

Synopsis

`double {number}`

Description

When given a number `number`, return its value cast as a double.

Important

This command does not appear in the MIDP1.0 version of Hecl, because it doesn't deal with floating point.

Name

eq — Tests string equivalence.

Synopsis

`eq {string1} {string2}`

Description

The **eq** commands compares two strings, returning 1 if they are equal, 0 if they are not.

Example

```
if {eq 1 1.0} {
    puts "True"
} else {
    puts "False"
}
```

Produces:

```
False
```

Despite being numerically equivalent, the strings "1" and "1.0" are different.

Name

eval — Evaluate Hecl code.

Synopsis

`eval {*code*}`

Description

The **eval** command takes a string containing Hecl commands, evaluates them, and returns the result.

Example

```
set i 0
set str {incr}
lappend $str "i"
eval $str
puts $i
```

Produces:

```
1
```

Name

exit — Exit the current process

Synopsis

`exit` [*`exitcode`*]

Description

The **exit** command stops the execution of the current process. It causes the *`exitcode`* (an integer) to be returned by the process.

Name

false — Return false

Synopsis

```
false
```

Description

The **false** command is the opposite of **true**, and always returns a false value.

Name

filter — Filter a list.

Synopsis

`filter` {*`list`*} {*`varname`*} {*`script`*}

Description

The **filter** command takes a list and filters it according to the code provided in *`code`*. The current element of the list being considered is stored in the *`varname`* provided. A list of 'matches' is returned.

Example

```
set lst {1 2 3 4 5 4 3 2 1}
puts [filter $lst x {= $x 4}]
```

Produces:

```
4 4
```

Name

float — Cast as a float

Synopsis

`float` {*`number`*}

Description

When given a number *`number`*, return its value cast as a float.

Important

This command does not appear in the MIDP1.0 version of Hecl, because it doesn't deal with floating point.

Name

for — For loop.

Synopsis

`for` {*`initialization`*} {*`test`*} {*`step`*} {*`body`*}

Description

The **for** command is like in many other languages like C and Java. As arguments, it takes an *`initialization`* option, which is often used to set a variable to some initial value, a *`test`* to determine whether to continue running, a *`step`* script option which is run at each iteration of the body (to increment a variable, for example), and the body itself.

Example

```
set out {}
for {set i 0} {< $i 10} {incr $i} {
    append $out $i
}
puts $out
```

Produces:

```
0123456789
```

Name

foreach — Iterate over elements in a list.

Synopsis

`foreach` {*`varname`*} {*`list`*} {*`body`*} `foreach` {*`varlist`*} {*`list`*} {*`body`*}

Description

The foreach command iterates over a list. For each element of the list, *`varname`* is set to a new element of the list, and then *`body`* is run.

Example

```
set lst {a b c d e}
set res {}
foreach el $lst {
    append $res $el
}
puts $res
```

Produces:

```
abcde
```

Name

global — Use global variable from within a proc.

Synopsis

`global` {*`varname`*} [*`varname`*...]

Description

By default, Hecl variables are always local. Global variables are not visible from within procedures. The **global** command makes global variable *`varname`* visible within a procedure.

Example

```
set foo 1
proc incfoo {} {
    global foo
    incr $foo
}
incfoo
puts $foo
```

Produces:

```
2
```

Name

hasclass — An interface to `Class.forName`

Synopsis

`hasclass` {*`classname`*}

Description

Reports whether a given class is present or not.

Important

Keep in mind that in J2ME, using an obfuscator, class names may not be what you think they are! Only use this for system-defined classes.

Example

```
hecl> hasclass org.hecl.net.HttpCmd
1
hecl> hasclass oogyboogy
0
```

Name

hash — Create and manipulate hash tables.

Synopsis

`hash` {*list*} `hget` {*hash*} {*key*} `hset` {*hash*} {*key*} {*value*} `hcontains` {*hash*} {*key*} `hclear` {*hash*} `hkeys` {*hash*} `hremove` {*hash*} {*key*}

Description

The **hash** command takes an even-numbered list and creates a hash table from it, using the even elements as keys, and odd elements as values. A new hash table is returned. The **hget** and **hset** commands operate on hash tables. Both take a hash table as their first argument. **hget** also takes a key, and returns the corresponding value, or an error if no key by that name exists. To determine whether a given key exists, use the **hcontains** command, which returns true or false depending on whether the key exists in the hash table.

The **hkeys** command returns the keys of the hash table, as a list.

The **hclear** command clears an entire hash table, whereas **hremove** removes the value associated with a given key.

Example

```
set foo [hash {a b c d}]
hset $foo a 42
puts [hget $foo a]
```

Produces:

```
42
```

Name

if — Conditionally execute code.

Synopsis

`if` {*`test`*} {*`code`*} [{`elseif`} | {*`test`*} | {*`code`*} ...] [{`else`} | {*`code`*}]

Description

The if command executes Hecl code conditionally. In its most basic form, it executes a *`test`*. If the results are not 0, then it executes *`code`*. If not, no further actions take place. **if** may take any number of `elseif` clauses, which have their own *`test`* and *`code`*. Finally, if none of the conditions has matched, it is also possible to supply an `else` clause that will be executed if the results of the if and elseif tests were all false.

Example

```
if { true } {
    puts "true"
} else {
    puts "false"
}
```

Produces:

```
true
```

```
if { > 0 1 } {
    puts "true"
} else {
    puts "false"
}
```

Produces:

```
false
```

Name

incr — Increment a variable.

Synopsis

`incr` {*`varreference`*} {*`integer`*}

Description

The **incr** command takes a variable reference, and adds *`integer to it`*.

Example

```
set foo 1
incr $foo
puts "foo is $foo"
incr $foo 10
puts "foo is now $foo"
```

Produces:

```
2
12
```

Name

intro — Introspection command.

Synopsis

`intro` [{commands}]

Description

The **intro** command is used for Hecl introspection. It takes a subcommand which causes it to perform the desired function.

Example

```
 puts [sort [intro commands]]
```

Produces: (depending on the available commands)

```
* + - / < = > append break catch continue copy eq eval filter for
foreach global hash hget hset if incr intro join lappend lindex
list llen lset proc puts ref return search set strindex strlen sort
source split time true upeval while
```

Name

join — Join elements of a list to create a string.

Synopsis

`join {list} [string]`

Description

The **join** command takes a *`list`* argument, and optionally, a *`string`* argument. It joins all elements of the list together with the string, or, if a string is not provided, with a space.

Example

```
puts [join {a b c} "|"]
```

Produces:

```
a|b|c
```

Name

lappend — Append an element to a list.

Synopsis

`lappend` {*`listreference`*} {*`element`*}

Description

The **lappend** takes a reference to a list, and an element to add to that list.

Example

```
set foo a
lappend $foo "b"
puts $foo
lappend $foo "c d"
puts $foo
```

Produces:

```
a b
a b {c d}
```

Name

lindex — Return the Nth element of a list

Synopsis

`lindex` {*`list`*} {*`index`*}

Description

The **lindex** command takes a list and an index number as arguments, and return's the index'th element of the list.

Example

```
puts [lindex {a b c} 2]
```

Produces:

```
c
```

Name

list — Create a list

Synopsis

`list` {*element*} [{*element*} ...]

Description

The **list** command takes any number of arguments and returns a list.

Example

```
puts [list a b c [list 1 2 3]]
```

Produces:

```
a b c {1 2 3}
```

Name

llen — List length.

Synopsis

`llen {list}`

Description

The **llen** returns the length of its list argument.

Example

```
puts [llen {1 2 3 {a b c}}]
```

Produces:

```
4
```

Name

lrange — Get range of elements from a list.

Synopsis

`lrange` {*`list`*} {*`first`*} {*`last`*}

Description

The **lrange** command fetches a range of elements from *`list`*, starting at element *`first`* and ending at *`last`*.

Example

```
lrange {a b c d e f g} 0 2
```

Produces:

```
a b c
```

Name

lset — Set list elements.

Synopsis

`lset` {*`listref`*} {*`index`*} [*`replacement`*]

Description

The **lset** command sets the *`index`*'th element of the list to *`replacement`*. If *`replacement`* is not present, then the element is deleted.

Example

```
    set lst {a b c}
    lset $lst 1 x
    puts $lst

    lset $lst 1
    puts $lst
```

Produces:

```
a x c
a c
```

Name

ne — String "not equal".

Synopsis

`ne` {*`string1`*} {*`string2`*}

Description

The **ne** commands compares two strings, returning 0 if they are equal, 1 if they are not.

Example

```
if {ne 1 00001} {
    puts "True"
} else {
    puts "False"
}
```

Produces:

```
True
```

Name

not — Logical not

Synopsis

`not` {*number*}

Description

The **not** command performs a logical not on the argument given to it.

Example

```
hecl> not 0
1
hecl> not 1
0
hecl> not 4
0
```

Name

or — Logical or

Synopsis

`or {number} [number][number][...]`

Description

The **or** command does a binary or of the numbers passed to it, so it can also be used as a logical or.

Example

```
hecl> or 1
1
hecl> or 0
0
hecl> or 1 0
1
hecl> or 2 4
6
hecl> or 1 2 4
7
```

Name

proc — Create a new procedure.

Synopsis

`proc` [*`name`*] {*`arglist`*} {*`body`*}

Description

The **proc** command creates new procedures, which are virtually indistinguishable from built-in Hecl commands. *`name`* is the name of the new command, or, if it is absent, an anonymous procedure is created (and should be stored in a variable). *`arglist`* is a list of arguments that the new command will take and make available as local variables within the *`body`*, which is the code executed every time the command is called. If the last element of the argument list is `args`, the variable `args` is a list that is filled with any arguments (including 0) above and beyond the number of arguments specified for the proc.

Example

```
proc addlist {lst} {
    set res 0
    foreach e $lst {
 incr $res $e
    }
    return $res
}

puts [addlist {1 2 3 4 5}]
```

Produces:

```
15
```

args Example

```
proc showargs {args} {
    puts "Args: $args"
}
showargs
showargs x y z
```

Produces:

```
Args:
Args: x y z
```

Anonymous proc Example

```
set foo [proc {x} { puts $x }]
```

```
$foo beebop
```

Produces:

```
beebop
```

Name

puts — Print text.

Synopsis

`puts {text}`

Description

The **puts** command prints *`text`* to stdout.

Example

```
puts "Hello, world"
```

Produces:

```
Hello, world
```

Name

rename — Rename a command

Synopsis

`rename` {*cmdname*}

Description

Renames a Hecl command.

Example

```
hecl> rename puts send_it_to_the_screen
hecl> send_it_to_the_screen "hello world"
hello world
```

Name

return — Returns a value from a procedure.

Synopsis

`return` {*value*}

Description

The **return** command returns a value from a **proc** command.

Example

```
proc someproc {} {
    set res 1
    return $res
    set res 2
    return $res
}
puts [someproc]
```

Produces:

```
1
```

Name

search — Find the first instance of something in a list.

Synopsis

`search {list} {varname} {script}`

Description

The **search** command is similar to **filter** in functionality, except that it stops searching on the first match.

Example

```
set lst {1 2 3 4 5 4 3 2 1}
puts [search $lst x {= $x 4}]
```

Produces:

```
4
```

Name

set — Set a variable.

Synopsis

`set` {*`varname`*} [*`value`*]

Description

The **set** sets the value of a variable *`varname`* to value *`value`*. If *`value`* is not provided, returns the value of *`varname`*.

Example

```
set foo "bar"
set bee bop
puts "foo is $foo and bee is $bee"
```

Produces:

```
1
foo is bar and bee is bop
```

Name

sort — Sorts list alphabetically.

Synopsis

`sort {list}`

Description

The **sort** command returns an alphabetically sorted list of the contents of *`list`*.

Example

```
puts [sort {padova rovigo verona vicenza venezia treviso belluno}]
```

Produces:

```
belluno padova rovigo treviso venezia verona vicenza
```

Name

split — Split a string into a list.

Synopsis

`split` {*`string`*} [*`splitstring`*]

Description

The **split** command takes a string and splits it into a list, divided by *`splitstring`*, which defaults to " " if not present.

Example

```
puts [split "aaa;bbb;ccc" ";"]
puts [split "aaa bbb ccc"]
puts [split "aaaxbbbycccxyddd" "xy"]
```

Produces:

```
aaa bbb ccc
aaa bbb ccc
aaaxbbbyccc ddd
```

Name

strbytelen — Return the length of the string, in bytes.

Synopsis

```
strbytelen {string}
```

Description

The **strbytelen** returns the length of `string`, in bytes. The number of characters and bytes may be different because of multi byte character encodings.

Name

strcmp — Compare two strings, return 0 if equal.

Synopsis

`strcmp {`*`stringA`*`} {`*`stringB`*`}`

Description

The **strcmp** takes two strings and compares them, returning 0 if they are equal, 1 if the first string is "greater than" the second string, or -1 if the first string is "less than" the second string.

Name

strfind — Find one string in another.

Synopsis

`strfind` {*`string1`*} {*`string2`*}

Description

The **strfind** looks for the first occurence of *`string1`* in *`string2`*. If it finds a match, it returns the index where the first letter of the match lies. If it is not found, it returns -1.

Name

strindex — Return the index'th character of string.

Synopsis

`strindex` {*`string`*} {*`index`*}

Description

The **strindex** command returns the *`index`*'th character of *`string`*.

Example

```
puts [strindex "Hello, world" 0]
puts [strindex "Hello, world" 11]
```

Produces:

```
H
d
```

Name

strlast — Get the last occurance of one string within another string

Synopsis

`strlast` {*`string`*} {*`string_to_search`*} [*`start_index`*]

Description

Search *`string_to_search`* for the last place that *`string`* occurs, and return the index. If *`start_index`* is provided, start searching from that position.

Name

strlen — String length.

Synopsis

`strlen {string}`

Description

The **strlen** returns the length of *`string`*.

Example

```
puts [strlen "abcdefghijklmnopqrstuvwxyz"]
```

Produces:

```
26
```

Name

strlower — Lower case a string

Synopsis

`strlower` {*`string`*}

Description

The **strlower** command returns a lower-cased version of *`string`*.

Name

strrange — Return a substring from a string

Synopsis

`strrange` {*`string`*} {*`start`*} {*`end`*}

Description

The **strrange** command returns a substring composed of the characters in *`string`* from positions *`start`* to *`end`*.

Name

strrep — Repeat a string N times

Synopsis

`strrep` {*`string`*} {*`times`*}

Description

The **strrep** command returns *`times`* copies of *`string`*.

Name

strreplace — Replace string A in string B

Synopsis

`strreplace` { {*{from to}*} } {*string*}

Description

The **strreplace** command replaces each instance of `from` with `to` in `string`.

Example

```
puts [strreplace {hi hello} blahhiblahblahhihiblah]
```

Produces:

```
blahhelloblahblahhellohelloblah
```

Name

strtrim — Remove whitespace or other characters from the beginning/end of a string

Synopsis

`strtrim` {*`string`*} [*`totrim`*] `strtriml` {*`string`*} [*`totrim`*] `strtrimr` {*`string`*} [*`totrim`*]

Description

The **strtrim** command "trims" any leading or trailing whitespace from the string passed to it. The **strtriml** and **strtrimr** commands trim from, and only from, the left and right sides of the string, respectively. All of the trim commands take an optional argument that specifies what exactly to trim.

Examples

```
hecl> strtrim " foo "
foo
hecl> strtrim "hello world" "he"
llo world
hecl> strtriml "xxxyyyzzzxxx" xxx
yyyzzzxxx
hecl> strtrim "        alone          "
alone
```

Name

strupper — Upper case a string

Synopsis

`strupper` {*`string`*}

Description

The **strupper** command returns a upper-cased version of *`string`*.

Name

system.hasproperty — Check for existence of system properties

Synopsis

`system.hasproperty` {*propertyname*}

Description

Query for the existence of system properties that can be retrieved with the **system.getproperty** command.

Name

system.getproperty — Get system properties

Synopsis

`system.getproperty {propertyname}`

Description

Retrieve information about system properties. Here is a list of some available system properties on Java ME platforms: http://developers.sun.com/mobility/midp/questions/properties/index.html

Name

throw — Generates an exception that can be caught with **catch**

Synopsis

`throw` {*`message`*} [*`errortype`*]

Description

The **throw** command generates an exception, that can be caught with the **catch command**. *`message`* is a human readable error message, whereas *`errortype`* is a code that can be matched by code doing a **catch**. It can be used as a way for Hecl programs to distinguish between different types of errors in a global catch section of code.

Examples

```
hecl> throw "oh no"
{ERROR {oh no}} {throw 1}
hecl> throw "oh no" USERERROR
{USERERROR {oh no}} {throw 1}
```

Name

time — Time the execution of a script.

Synopsis

`time` {*`script`*} [*`repetitions`*]

Description

The **time** command executes *`script`* *`repetitions`* times, if *`repetitions`* is present, or once if not. It measures the amount of time taken by this execution in milliseconds, and returns it.

Example

```
set time [time {
    set i 0
    while { < $i 100 } {
 incr $i
    }
} 10]
puts "Time is $time"
```

Produces:

```
Time is 6
```

Name

true — Returns true.

Synopsis

```
true
```

Description

The **true** command returns 1, or true.

Example

```
if { true } {
    puts "true is true"
}
```

Produces:

```
true is true
```

Name

unset — Unset a variable.

Synopsis

`unset` {*varname*}

Description

The **unset** command unsets a variable.

Name

upeval — Evaluate script in next stack frame up.

Synopsis

```
upeval[level]{script}
```

Description

The **upeval** command evaluates *script* one stack frame up from the current stack frame, if *level* is not present. If *level* is present, **upeval** behaves like so: if *level* is less than one, the *script* is run that many levels down from the top of the stack. If it's 0, the script is run at the global level, and if it's a positive number, the script is run at that absolute level up from the global namespace.

Example

```
proc stackframe {} {
    upeval { incr $foo }
}
set foo 1
stackframe
puts $foo
```

Produces:

```
2
```

Name

while — While loop.

Synopsis

`while` {*`condition`*} {*`body`*}

Description

The **while** command continues to evaluate *`body`* while *`condition`* is true.

Example

```
set i 0
while { < $i 6 } {
    puts "i is $i"
    incr $i
}
```

Produces:

```
i is 0
i is 1
i is 2
i is 3
i is 4
i is 5
```

Hecl Extension Commands

These commands are available in various Hecl extensions, that may or may not be available in different environments.

File Interaction

J2ME systems do not always have a file system, so these are not necessarily available. It is possible to compile Hecl with or without these commands.

Note

The **filefinder** command is a bit different from the others, in that it is a graphical widget that is used to select files, rather than a command that only interacts with the filesystem.

Name

filefinder — File selection widget.

Synopsis

`filefinder` [-startdir *`start_directory`*] [-selectedcmd *`selected_command`*] [-matchcmd *`match_command`*] [-errorcmd *`error_command`*]

Description

Creates a **lcdui.list** based file selection dialog that lets the user navigate up and down through directories and select a file. The options are as follows:

- `-startdir`: selects the directory to start the search in.
- `-selectedcmd`: a **proc** that by default takes one argument: the name of the selected file.
- `-matchcmd`: a **proc** that takes one argument: the full URL of the file to match against. If the file is a match, the proc must return a 1, otherwise 0. If a match occurs, the `-selectedcmd` will subsequently be called.
- `-errorcmd`: a **proc** that is called with one argument: an error message. This is called when some error occurs and can be used to display an error message.

Name

open — Opens a file for reading or writing.

Synopsis

`open` {*`filename`*} [w]

Description

Opens `filename` for reading or writing, depending on whether the `w` flag is set or not. The form of `filename` is platform-dependant. On Java ME systems, it needs to be a URI.

Name

source — Evaluate Hecl script in a file.

Synopsis

`source {filename}`

Description

The **source** command evaluates the Hecl script located in file *`filename`*.

Example

```
# Variable foo is defined as "Hello world" in foo.hcl
source foo.hcl
puts $foo
```

Produces:

```
Hello world
```

Name

file.basename — Returns the file name with no paths.

Synopsis

`file.basename` {*`filename`*}

Description

Returns the file's name, with no URL schema or paths.

Example

```
file.basename file:///foo/bar/bee/bop.txt
```

Would return

```
bop.txt
```

Name

file.cd — Change working directory.

Synopsis

`file.cd` {*`directory`*}

Description

The **cd** command changes the current working directory to *`directory`*. Or at least it changes where *Java* thinks the working directory is. It does not change the process' actual working directory. Apparently that's not possible with Java.

Name

file.current — Returns the name of the file currently being evaluated.

Synopsis

```
file.current
```

Description

Returns the name of the file currently being evaluated. For instance, if a file `foobar.hcl` is being sourceed, and there is a call to **file.current** in it, it will return the name of the file, `foobar.hcl`.

Name

file.delete — Delete the file.

Synopsis

`file.exists` {*`filename`*}

Description

Delete `filename`.

Name

file.devs — List available devices.

Synopsis

```
file.devs
```

Description

Returns a list of the root devices.

Name

file.du — "Disk" usage.

Synopsis

`file.du` {*`filename`*}

Description

Returns a hash with the following keys: `total` - the total amount of space on the storage device where the file resides, and `used` - the amount of space currently utilized.

Name

file.exists — Does the file exist?

Synopsis

`file.exists` {*`filename`*}

Description

Returns 1 if the file exists, otherwise 0.

Name

file.hidden — Is the file hidden?

Synopsis

`file.hidden` {*filename*}

Description

Return 1 if the file is "hidden". To quote from the relevant Java documentation:

> The exact definition of hidden is system-dependent. For example, on UNIX systems a file is considered to be hidden if its name begins with a period character ('.'). On Win32 and FAT file systems, a file is considered to be hidden if it has been marked as such in the file's attributes. If hidden files are not supported on the referenced file system, this method always returns false.

Name

file.isdirectory — Is this a directory?

Synopsis

`file.isdirectory` {*path*}

Description

Returns 1 if the path in question is a directory, otherwise 0.

Name

file.isopen — Is the file currently in use?

Synopsis

`file.isopen` {*`filename`*}

Description

Returns 1 if the file is currently open, otherwise 0.

Name

file.join — Creates a filename from a list.

Synopsis

`file.join {list of path elements}`

Description

Given a list of path elements, like {foo bar bee bop}, creates a filename like foo/bar/bee/bop.

Name

file.list — List files in a directory.

Synopsis

`file.list` {*`directory`*}

Description

Lists the files in the specified directory.

Name

file.mkdir — Creates the directory.

Synopsis

`file.mkdir` {*directory*}

Description

Creates the named directory.

Name

file.mtime — Returns the file modification time.

Synopsis

`file.mtime` {*`filename`*}

Description

Returns the time the file was last modified, as a value expressed in milliseconds since the "epoch" (00:00:00 GMT, January 1, 1970).

Name

file.readable — Is the file readable?

Synopsis

`file.readable {`*`filename`*`}`

Description

Checks to see whether the file exists and is readable, and if so, returns 1. Otherwise, returns 0.

Name

file.rename — Rename a file.

Synopsis

`file.rename` {*`oldname`*} {*`newname`*}

Description

Renames `oldname` to `newname.`

Name

file.size — Returns the file's size.

Synopsis

`file.size` {*`filename`*}

Description

Returns `filename`'s size, in bytes.

Name

file.split — Return a list of a path's components.

Synopsis

`file.split` {*filename*}

Description

Takes a filename and returns a list of all the file's components.

Name

file.truncate — Truncates the file.

Synopsis

`file.truncate` {*`filename`*} {*`newlength`*}

Description

Truncates the file, making it at most `newlength` bytes long.

Name

file.writable — Is the file writable?

Synopsis

`file.writable {filename}`

Description

Checks to see whether the file exists and is writable, and if so, returns 1. Otherwise, returns 0.

HTTP

Even basic J2ME-enabled cell phones can access web pages (although they may not be able to create TCP/IP sockets). Hecl provides basic http commands

Name

http.geturl — Fetch contents of a URL.

Synopsis

`http.geturl` {*`url`*} [-query *`POSTdata`*] [-headers *`list of headers`*] [-validate [1|0]]

Description

The **http.geturl** command performs an HTTP request to the given *`url`* and returns information about the results. This information is returned as a hash table that can be accessed with the various h* commands such as **hget**. The list of keys returned includes the following, which are set by Hecl. Also created as keys are the various HTTP headers, such as connection, content-length, content-type, last-modified, and so on.

- `binary`: 1 if the data returned is binary, otherwise 0.
- `charset`: The charset used for the returned data.
- `data`: The returned data. For instance, if you fetched an ordinary web page, this would be its HTML contents.
- `ncode`: The numeric code of the response, such as 404, 500, 301, and so on.
- `status`: The request status - ok if the request was successfully processed.

The `-query` option is used to send key/value pairs of variables. In order to set random headers, the `-headers` argument is used. The `-validate` option exists to send a `HEAD` request.

Example

```
# Evaluate Hecl script located on a web site.
eval [hget [http.geturl http://www.hecl.org/somescript.hcl] data]
```

Add some headers to a request:

```
http.geturl http://www.hecl.org/ -headers {Set-Cookie foo=bar}
```

Name

http.formatQuery — URL Encode a request

Synopsis

`http.formatQuery [key val]`

Description

The **http.formatQuery** command takes a series of key value pairs and urlencodes them. Useful in order to pass data to the `-query` option of **http.geturl**

Name

http.data — A shortcut to get the data from an http.geturl response.

Synopsis

`http.data` {*request*}

Description

The **http.data** command is equivalent to an **hget** with `data` as the key.

```
http.data [http.geturl http://www.dedasys.com]
```

Name

http.ncode — A shortcut to get the numeric code from an http.geturl response.

Synopsis

`http.ncode` {*request*}

Description

The **http.data** command is equivalent to an **hget** with `ncode` as the key.

```
http.ncode [http.geturl http://www.dedasys.com]
```

Name

http.status — A shortcut to get the status from an http.geturl response.

Synopsis

`http.status {request}`

Description

The **http.status** command is equivalent to an **hget** with `status` as the key.

```
http.status [http.geturl http://www.dedasys.com]
```

Location

Access to your phone's location API (JSR 179). Get information on your precise location.

Note

Not available on all phones.

Name

location.get — Get location information.

Synopsis

`location.get` { {*`timeout`*} | { {-callback} {*`callbackProc`*} [{-timeout} {*`timeout`*}] [{-onerror} {*`onErrorProc`*}] } }

Description

The **location.get** command operates in two ways:

1. The simplest way is to simply call the command with a `timeout` argument. The program will block until the command returns an answer, in the form of a hash table (see below), which could take up to several minutes.

2. The alternative way of calling **location.get** is to pass it a `-callback` *`callbackProc`*, which is the name of a proc that will be called when the information is available, with a hash table specifying the results. Program execution continues and does not block on the location.get call. The `callbackProc` takes one argument, which is a hash of information about the location. When using `-callback` it is also possible to specify a `-timeout` option (in seconds), and an `-onerror` option. This is used in case of errors: the specified *`onErrorProc`*

Either method will supply the user with a hash table with the following elements. See also the JavaDocs here: javax/microedition/location/Location [http://library.forum.nokia.com/index.jsp?topic=/Java_Developers_Library/GUID-4AEC8DAF-DDCC-4A30-B820-23F2BA60EA52/javax/microedition/location/Location.html]

- `lat`: latitude.
- `lon`: longitude.
- `alt`: altitude.
- `haccuracy`: horizontal accuracy, in meters.
- `vaccuracy`: vertical accuracy, in meters.
- `location_method`: contains a hash table of its own with information about the method used to ascertain the location. Possible values include:
  - `ASSISTED`: Location method is assisted.
  - `UNASSISTED`: Location method is unassisted.
  - `NETWORKBASED`: Location is derived from the network.
  - `TERMINALBASED`: Location is obtained from a GPS terminal.
  - `ANGLEOFARRIVAL`: Location method Angle of Arrival for cellular / terrestrial RF system.
  - `CELLID`: Location method Cell-ID for cellular (in GSM, this is the same as CGI, Cell Global Identity).
  - `SATELLITE`: Location method using satellites (for example, Global Positioning System (GPS)).

    - `SHORTRANGE`: Location method Short-range positioning system (for example, Bluetooth LP).
    - `TIMEDIFFERENCE`: Location method Time Difference for cellular / terrestrial RF system (for example, Enhanced Observed Time Difference (E-OTD) for GSM).
    - `TIMEOFARRIVAL`: Location method Time of Arrival (TOA) for cellular / terrestrial RF system.
- `speed`: ground speed, in meters per second.
- `course`: course, in degrees relative to true north.

Net

Networking commands

Name

base64::decode — Base 64 decode

Synopsis

`base64::decode` {*`encoded-data`*}

Description

The **base64::decode** command does a base64 decode of the string passed to it.

Name

base64::encode — Base 64 encode

Synopsis

`base64::encode {data}`

Description

The **base64::encode** command does a base64 encode of the string passed to it.

RecordStore

Record Store is the name for persistent storage in J2ME. Even simple MIDP1.0 phones have this feature, and utilize it to store data between invocations of the application.

Name

rms.list — List available record store names, or id's for a name.

Synopsis

`rms.list` [*`rsname`*]

Description

The **rms.list** command, when called without arguments, returns a list of names of the available record stores that have already been created. When called with an *`record store name`* returns the list of id's that are currently in use for that record store.

Example

```
foreach name [rs.list] {
    puts "Listing of id's for $name :"
    puts [rs.list $name]
}
```

Produces:

```
yyy
```

Name

rms.size — Returns the size occupied by the record store

Synopsis

```
rms.size {rsname}
```

Description

The **rms.size** command returns the size, in bytes, occupied by the recordstore specified.

Name

rms.sizeavail — Returns the amount of space available to grow for this record store.

Synopsis

```
rms.sizeavail {filename}
```

Description

The **rms.sizeavail** command returns the size in bytes left to grow for the specified recordstore.

Name

rms.set — Sets the value of a record store

Synopsis

`rms.set` {*`rsname`*} [*`recordid`*] {*`data`*}

Description

The **rms.set** command takes, as arguments, the name of the record store; optionally, the record id to set, and finally, the data to insert into the record store. If the record id is not specified, the default value of 1 is used.

Name

rms.get — Fetch the value of the specified record store and record id

Synopsis

`rms.get` {*`rsname`*} [*`recordid`*]

Description

The **rms.get** command returns the value of the data stored in record store *`rsname`*. The optional argument *`recordid`* specifies the record id, and defaults to 1. An error is thrown if the record does not exist.

Example

```
rms.set highscore 1 [highscore]
... application restarted ...
set highscore [rms.get highscore 1]
```

Name

rms.add — Adds data to the specified record store, returning the new record id.

Synopsis

```
rms.add {data}
```

Description

The **rms.add** command adds `data` to the named record store, returning the record id with which it may be retrieved in the future.

Name

rms.delete — Deletes either the record store, or a record id associated with a record store.

Synopsis

`rms.delete` {*rsname*} [*recordid*]

Description

The **rms.delete** command deletes either the entire record store, if a *`record id`* is not specified, or if one is, the record id in question.

Name

rms.hset — Sets a key/value pair in the given record store.

Synopsis

`rms.hset` {*`rsname`*} {*`key`*} {*`value`*}

Description

The **rms.hset** command, given the name of a record store *`rsname`* treats it as a hash table, setting the value of *`key`* with *`data`*.

Name

rms.hget — Gets the value of a given key in the specified record store

Synopsis

`rms.hget {rsname} {key}`

Description

The **rms.hget** command returns the data associated with *`key`* in the named record store.

Name

rms.hexists — Determine whether a given key exists in a record store.

Synopsis

```
rms.hexists {rsname} {key}
```

Description

The **rms.hexists** command returns 1 if the given `key` exists in the named record store.

Name

rms.hkeys — Returns all keys associated with a given record store.

Synopsis

`rms.hkeys {`*`rsname`*`}`

Description

The **rms.hkeys** command returns a list of keys associated with the named record store.

Name

rms.hdel — Deletes a key from the specified record store.

Synopsis

`rms.hdel` {*`rsname`*} {*`key`*}

Description

The **rms.hdel** command deletes *`key`* from record store *`rsname`*.

K-XML

The K-XML extension provides the kXML 2 parser functionality for hecl. kXML 2 implements the XmlPull API. Please find general information about XmlPull parsers including the interface documentation at xmlpull.org.

Note

Kxml is not compiled into the default build, so you have to change the kxml property from 0 to 1 in the `cldc11midp20.properties` file and recompile Hecl.

Name

kxml.create — Returns the kxml parser object.

Synopsis

```
kxml.create
```

Description

The **kxml.create** creates a kxml parser object.

```
set xmlParser [kxml.create]
```

Name

kxml.gettext — Returns the text content of the current event as a string.

Synopsis

```
kxml.gettext
```

Description

The **kxml.gettext** returns the text content of the current XML event as a string.

The value returned depends on current event type, for example for TEXT event it is element content (this is typical case when **kxml.next** is used). See description of **kxml.nextToken** for detailed description of possible returned values for different types of events.

Note

In case of ENTITY_REF, this method returns the entity replacement text (or an empty string if not available).

Name

kxml.input — Sets a input stream the kxml-parser is going to process.

Synopsis

`kxml.input` { *`xmlParser`*} { *`xmlStream`*}

Description

Set the xml input stream to the given xmlParser.

```
set http [http.geturl "http://www.google.com/ig/api?weather=$city,$country"]
set xmlStream [hget $http data]
kxml.input $xmlParser $xmlStream
```

Name

kxml.nexttag — Call **kxml.next** and return event if it is START_TAG or END_TAG otherwise throw an exception.

Synopsis

`kxml.nexttag` { *`xmlParser`*}

Description

Call **kxml.nexttag** and return event if it is START_TAG or END_TAG otherwise throw an exception.It will skip whitespace TEXT before actual tag if any.

Name

kxml.next — Get next parsing event.

Synopsis

`kxml.next` { *`xmlParser`*}

Description

The **kxml.next** advance the parser to the next event. The int value returned from next determines the current parser state and is identical to the value returned from following calls to **kxml.event**.

The following event types are seen by **kxml.next** .

- `START_DOCUMENT` `0` :Initially, the parser is in the START_DOCUMENT state.
- `START_TAG` `2`: An XML start tag was read.
- `TEXT` `4`: Text content was read; the text content can be retreived using the **kxml.gettext** method.
- `END_TAG` `3`: An end tag was read.
- `END_DOCUMENT` `1`: No more events are available.

Name

kxml.requirestart — Test if the current event is START_TAG type and do match the given name.

Synopsis

`kxml.requirestart` { *`xmlParser`* } { *`name`* }

Description

Test if the current event is START_TAG type do match the given name and any namesapce. If the test is not passed, an exception is thrown. The exception text indicates the parser position, the expected event and the current event that is not meeting the requirement.

```
# assert the current event is a START_TAG event "forecast_conditions"
kxml.requirestart $xmlParser "forecast_conditions"
```

Name

kxml.requireend — Test if the current event is END_TAG type and do match the given name.

Synopsis

`kxml.requireend` { *`xmlParser`*} { *`name`*}

Description

Test if the current event is END_TAG type and do match the given name and any namespace. If the test is not passed, an exception is thrown. The exception text indicates the parser position, the expected event and the current event that is not meeting the requirement.

```
# assert we have reach the END_TAG event of "forecast_conditions"
kxml.requireend $xmlParser "forecast_conditions"
```

Name

kxml.attrcount — Returns the number of attributes of the current start tag.

Synopsis

`kxml.attrcount { xmlParser}`

Description

Returns the number of attributes of the current start tag, or -1 if the current event type is not START_TAG.

Name

kxml.attrvalue — Returns the given attributes value.

Synopsis

`kxml.attrvalue` { *`xmlParser`* } { *`index`* }

Description

Returns the given attributes value.

Name

kxml.getname — For START_TAG or END_TAG events, the name of the current element is returned.

Synopsis

`kxml.getname` { *`xmlParser`*}

Description

For START_TAG or END_TAG events, the name of the current element is returned.

```
# loop over the stream until we reach the END_TAG event "current_conditions"
while {!= [strcmp [kxml.getname $xmlParser] "current_conditions"] 0 } {
    set name [kxml.getname $xmlParser]
    puts $name
    kxml.nexttag $xmlParser
}
```

Interfacing Hecl and Java

Hecl is not a replacement for Java, and is indeed meant to work hand in hand with Java. We attempt to make it as easy as possible to call Java from Hecl, via the creation of new Hecl commands that can call Java code, in addition to calling Hecl from Java, which is a matter of a few lines of code.

Calling Hecl code from Java

```
import org.hecl.files.HeclFile;

import org.hecl.Eval;
import org.hecl.Interp;
import org.hecl.Thing;
import org.hecl.ListThing;
import org.hecl.HeclException;

...

 try {
     /* First, create a new interpreter, and pass it a
      * mechanism to load resources with - in this case,
      * files. */
     Interp interp = new Interp();

     /* Add the files package  */
     new HeclFile().loadModule(interp);
     /* Evaluate the file at args[0] */
     HeclFile.sourceFile(interp, args[0]);
     /* Evaluate some code in a string.  */
     String helloworld = new String("puts {Hello, world!}");
     interp.eval(new Thing(helloworld));
 } catch (Exception e) {
     System.err.println(e);
 }
```

The above code first creates a new interpreter. Next, it instantiates the HeclFile system. This isn't part of the Hecl core, because some systems, like J2ME, may not have files. If you don't have files, you can still use `interp.eval` to evaluate some code, which could come from whatever source you desire.

Creating new Hecl commands

Creating new Hecl commands is relatively simple. The first step is to create a new class for your command in a file, say `HelloCmd.java`, which, for simplicity's sake, you could put in `core/org/hecl/`. The code would look something like this:

```
import org.hecl.Command;
import org.hecl.HeclException;
import org.hecl.Interp;
import org.hecl.Thing;

class HelloCmd implements Command {

    public Thing cmdCode(Interp interp, Thing[] argv)
 throws HeclException {

 System.out.println("Hello world");
 return null;
    }
}
```

The command takes an interpreter and an array of `Thing`s as arguments, where the first `Thing` is the name of the command itself, and the others are the arguments to it.

The "glue" that connects the name of your Hecl command with the Java code is also relatively simple:

```
 interp.addCommand("hello", new HelloCmd());
```

The above code would be included somewhere in `commandline/Hecl.java`, `midp20/Hecl.java` or elsewhere, depending on what platform you're working on; or `core/org/hecl/Interp.java` if you want to include it as part of the Hecl core.

Easy, no? There are a few other useful methods that you should be aware of, to share variables between Hecl and Java, and to return results from your Hecl commands:

- `interp.setVar(`
  `Thing`
  `varname`
  `,`
  `Thing`
  `value`

```
    );
```

This sets the value of *`varname`* to some value.

-
  ```
  Thing interp.getVar(
       Thing
       varname
     );


  Thing interp.getVar(
       String
       varname
     );
  ```

  These methods take a variable name, either in string form or as a `Thing`, and return the `Thing` associated with that variable.

- `interp.result` is used to set the result of a command. This oft-used variable is accessed directly for simplicity, speed and smaller code size.

-
  ```
  int IntThing.get(
       Thing
       thing
     );
  ```

  Get an int from a Thing.

-
  ```
  String StringThing.get(
       Thing
       thing
     );
  ```

  Get a String from a Thing.

-
  ```
  Thing IntThing.create(
       int
       i
     );
  ```

  Creates a new thing from an int.

There are similar methods for strings, floats (where applicable), hashes and lists.

Building Hecl: Ant Targets

Since Hecl can be built for many different platforms, with many different capabilities, it requires a fairly complex build system, based on Ant [http://ant.apache.org/], Antenna [http://antenna.sourceforge.net/], and for Java ME, Proguard. [http://proguard.sourceforge.net/]. However, it's not necessary to understand the intricacies of this system unless you're doing more advanced hacking: in that case, you should ask on the mailing list if you need help.

The most important build targets are the following:

ant build: Build everything. This is a time-consuming process, but is the best way to make sure you have built everything with the latest code.

ant packageCommandline: Builds the `jars/j2se/Hecl.jar` file for the j2se/command line version of Hecl.
ant -propertyfile ./cldc11midp20.properties midlet: Builds the MIDP 2.0 version of Hecl.
ant -propertyfile ./cldc10midp10.properties midlet: Builds the MIDP 1.0 version of Hecl.
ant installAndroidPackage: Builds the Android version of Hecl and installs it on the Android emulator, which must be running for the final step to work.
ant docs: Creates both the DocBook documentation and runs Javadoc.

JavaDocs

For the complete Hecl javadoc documentation, see the Hecl Javadocs [jdocs]. And, of course, look at the Hecl source code to see how it's done!

Hecl and Java ME

Hecl is designed to be small enough to run on mobile devices such as cell phones. This means that for the Hecl core, has been necessary to limit ourselves to Java API's that work with J2ME.

Hecl Java ME Tutorial

Note

This is a tutorial showing you how to use Hecl to write applications for Java ME. If you want a simple introduction to the Hecl language, you can find that here: tutorial.

This tutorial first appeared here: Create a simple application with Hecl - Introducing Hecl, a mobile phone scripting language [http://www.freesoftwaremagazine.com/articles/creating_a_simple_application_with_hecl]. It is a tutorial style introduction writing Hecl code for mobile phones, by David Welton.

The aim of this tutorial is to help you create cell phone applications, so let's get started right away. You'll need a few things first:

- Sun's Java. With Ubuntu, you can get this like so:

  ```
  apt-get install openjdk-6-jdk
  ```

- Sun's WTK toolkit [http://java.sun.com/products/sjwtoolkit/index.jsp]. While you don't need the tools to compile Hecl (unless you want to hack on it!), you do want the emulator, so that you don't have to load your app onto your phone each time you want to test it. It's not open source software (yet?), but it does run on Linux, Mac and Windows, and of course it is free.

- Hecl itself. You can get the latest code here: http://www.hecl.org/downloads/hecl-latest.tgz.

  ### Note

  Hecl is always improving, so you should also consider checking out Hecl directly from git: **`git clone git://github.com/davidw/hecl.git`**

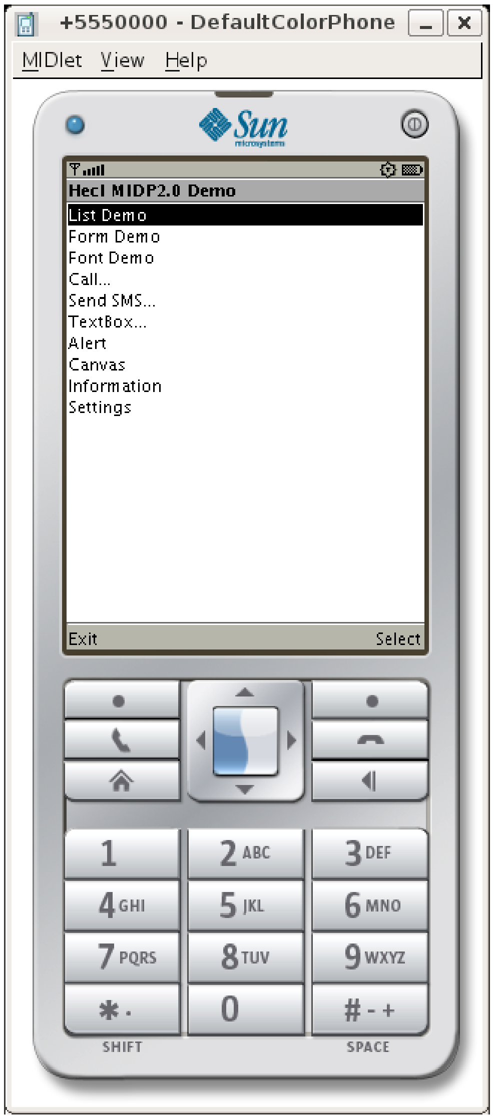
Sun's WTK requires installation - you can put it somewhere like `/opt`, so it won't get mixed up with the rest of your system. The installation process is very simple - just say yes to a few questions, and you're done. Hecl doesn't require installation: everything you need is already there in the distribution.

To see if everything's working, you can try launching the emulator with the sample application: **`/opt/WTK2.5.2/bin/emulator -classpath jars/cldc1.1-midp2.0/Hecl.jar Hecl`**

Note

With version 3 of the Sun WTK (which as of 2010-01 only runs on Windows and Mac), the command line you need is as follows: **`/opt/WTK2.5.2/bin/emulator -Xjam:install jars/cldc1.1-midp2.0/Hecl.jad`**

That should bring up something like this:



Hecl MIDP2.0 Demo
List Demo
Form Demo
Font Demo
Call...
Send SMS...
TextBox...
Alert
Canvas
Information
Settings
Exit
Select

Figure 1: Hecl demo screen shot

This is Hecl's built in demo - its source code is located in `midp20/script.hcl`, but before I get too far ahead of myself, let's go back and create the classic "Hello World" application, just to get started and see how to work with Hecl.

Note

Hecl actually comes in several flavors, with slightly different GUI commands - MIDP1.0 (older phones), which has fewer commands and doesn't do as much, and MIDP2.0, for newer phones, which has a lot more features. This tutorial utilizes the MIDP2.0 commands, because that's what current phones are based on. The concepts described are very similar for the MIDP1.0 commands, but the commands are slightly different. Please contact us if you are interested in a MIDP1.0 version of this tutorial.

To write your first Hecl program, open a text editor, and type the following program into a file - I'll call it `hello.hcl`:

```
proc HelloEvents {cmd form} {
    [lcdui.alert -text "Helllllllloooo, world!" -timeout forever] setcurrent
}

set form [lcdui.form -title "Hello world" -commandaction HelloEvents]
set cmd [lcdui.command -label "Hello" -longlabel "Hello Command" -type screen]

$form setcurrent
$form addcommand $cmd

$form append [lcdui.stringitem -label "Hello" -text "World"]
```

Not bad - 8 lines of code, and most of it's pretty clear just from looking at it. I'll go through it line by line, so you understand exactly what's happening.

1. The first bit of code, that starts with `proc HelloEvents`, defines a "procedure": in other words a function called **HelloEvents**. When this function is called, it creates an "alert" - think of it as a pop up message telling you something important. `-timeout forever` tells the message to stay on the screen until the user dismisses it.

2. The second command defines a form, with the command **lcdui.form**, with the title of "Hello World", and connected to the **HelloEvents** proc. What this connection means is that when any commands associated with the form are activated by the user, this procedure is called to handle them. The code `set form` stores the form object in the variable `form`, so that it can be referenced later.

3. The following line creates a command that can be activated by the user. It has the label "Hello", and is stored in the variable `cmd`. I use the `screen` type for the command, which is used for user defined commands. There are some other predefined types such as `exit` and `back`.

4. `$form setcurrent` references the previously created form, and tells Hecl to display it on the screen.

5. The **addcommand** subcommand (you could also think of it as a "method", like in an object oriented language) attaches the command I created above to the form. This makes the command visible in the form.

6. Finally, I display a string on the form with the **lcdui.stringitem** command. On most phones, the `-label` text is displayed in bold, and the `-text` text is displayed next to it.

That's it! Now, to transform the code into a cell phone application, run a command:

```
java -jar jars/JarHack.jar -hecljar jars/cldc1.1-midp2.0/Hecl.jar \
    -destdir ~/ -name Hello -script hello.hcl
```

This is all it takes - this command takes the existing `Hecl.jar` file, and replaces the Hecl script within with our newly created `hello.hcl` script, and creates the resulting `Hello.jar` in your home directory (referenced as `~/` in the command above).

Now, we can run the code in the emulator to see the application:

```
/opt/WTK2.5.2/bin/emulator -cp ~/Hello.jar Hecl
```

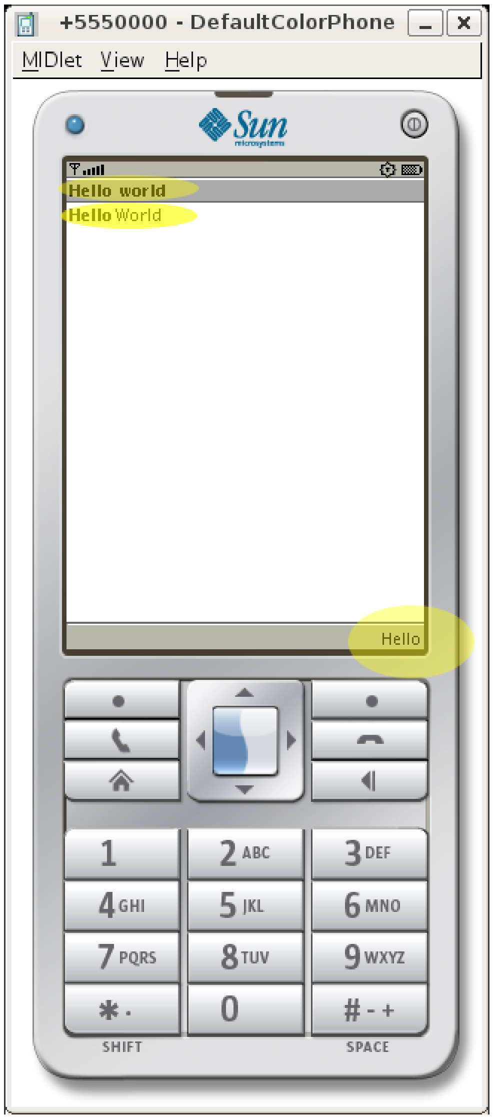


Figure 2: Hecl Hello World screenshot

Highlighted, from the top, are the form's `-title`, the **stringitem**, and in the lower right corner, the command labeled Hello.

If you press the "hello" button, the code in **HelloEvents** is executed, and an "alert" is popped up onto the screen, and stays there until you hit the "Done" button.

While creating an application is very easy, unfortunately, installing it on a phone is not; there isn't much that Hecl can do to ease that process, which is different for each phone. On Linux, for my Nokia telephone, I use the gammu program to transfer programs to my phone, like so:

```
gammu nothing --nokiaaddfile Application Hecl
```

Another method that may work better across different phones is to use the phone's browser to download and install the application, by placing the `.jar` and `.jad` files on a publicly accessible web server, and accessing the `.jad` file.

Note

Note that this will likely cost money in connection charges!

So far so good. Next, I'll create a small application that you can interact with to do something useful. It's a simplified version of the shopping list that can be found here [http://shoplist.dedasys.com]. The theory of operation behind this application is simple: typing a shopping list into a mobile phone is pretty painful - it's much better to do the data entry via a web page, and then fetch the list with the mobile phone application.

For this tutorial, I've created a simple list on the ShopList web site, with the PIN number `346764`, which can be viewed here [http://shoplist.dedasys.com/list/show/346764]. Feel free to create your own shopping lists - the site costs nothing to use. The cell phone application works like so: by entering the PIN, it downloads the list of items and displays them on the phone screen as a series of checkboxes. Have a look at the code to do this:

```
# Process events associated with the shopping list screen.
proc ShopListEvents {exitcmd backcmd cmd shoplist} {
    if { eq $cmd $exitcmd } {
 midlet.exit
    } elseif { eq $cmd $backcmd } {
 global shopform
 $shopform setcurrent
    }
}

# Create a new shopping list screen and fetch .
proc MakeList {exitcmd backcmd pin} {
    set url "http://shoplist.dedasys.com/list/fetch/${pin}"
    # Fetch the data, and retrieve the data field from the results hash.
    set data [hget [http.geturl $url] data]
    if { eq $data "PIN NOT FOUND" } {
 [lcdui.alert -type warning \
      -title "Pin Not Found" \
      -timeout forever\
      -text "The PIN $pin was not found on shoplist.dedasys.com"] setcurrent
 return
    }
    set shoplist [lcdui.list -title "Shopping List" \
        -type multiple]
    foreach e [split $data \n] {
 $shoplist append $e
    }
    $shoplist addcommand $exitcmd
    $shoplist addcommand $backcmd
    $shoplist setcurrent
    $shoplist configure -commandaction \
 [list ShopListEvents $exitcmd $backcmd]
}

# Process events associated with the main form.
```

```
proc ShopFormEvents {backcmd exitcmd pinfield
       fetchcmd cmd shopform} {
    if { eq $cmd $exitcmd } {
 midlet.exit
    } elseif { eq $fetchcmd $cmd } {
 MakeList $exitcmd $backcmd \
     [$pinfield cget -text]
    }
}

# The action starts here...

# Create a generic back command.
set backcmd [lcdui.command \
   -label Back \
   -longlabel Back -type back -priority 1]
# Create an exit command.
set exitcmd [lcdui.command \
   -label Exit \
   -longlabel Exit -type exit -priority 2]

# Create the form.
set shopform [lcdui.form -title "Shopping List"]
set pinfield [lcdui.textfield \
    -label "shoplist.dedasys.com PIN:" \
           -type numeric]
set fetchcmd [lcdui.command -label "Fetch" \
    -longlabel "Fetch Shopping List" \
    -type screen -priority 1]

$shopform append $pinfield
$shopform addcommand $exitcmd
$shopform addcommand $fetchcmd
$shopform setcurrent

$shopform configure -commandaction \
    [list ShopFormEvents $backcmd $exitcmd $pinfield $fetchcmd]
```

This is certainly more complex than the first example, but the general pattern is the same - screen widgets and items are created, displayed, and procs are called to deal with commands.

As I mentioned previously, commands with specific, predefined tasks have their own types, as I can see with the **back** and **exit** commands, which are respectively of types "back" and "exit".

After the two commands are defined, I create a form and add a textfield to it. By specifying `-type numeric` for the textfield, I indicate that it is only to accept numbers - no letters or symbols.

After creating the Fetch command, I append the textfield to the form (or else it wouldn't be visible), add the two commands to the form, and then, with **setcurrent**, make the form visible. The last line of code configures the form to utilize the **ShopFormEvents** proc to handle events. The list argument warrants further explanation:

Hecl, like many programming languages, has a **global** command that could be used in the various procs that utilize the **back** and **exit** commands - you could simply say `global backcmd`, and then the `$backcmd` variable would be available in that procedure. However, using global variables all over the place gets kind of messy, so what I want to do is pass in everything that the proc might need, and I do so by creating a list: `ShopFormEvents $backcmd $exitcmd $pinfield $fetchcmd`. You can see that these corresponds to the arguments that the proc takes: `proc ShopFormEvents {backcmd exitcmd pinfield fetchcmd cmd shopform}`, except for the last two, which Hecl *automatically* passes in. `cmd` is the command that was actually called, and `shopform` is of course the form that the proc was called with. By comparing `$cmd` with the various commands that are available, it's possible to determine which command called the proc, and act accordingly.

Now, let's build it and run it:

```
java -jar jars/JarHack.jar -hecljar jars/cldc1.1-midp2.0/Hecl.jar \
    -destdir ~/ -name ShopList -script shoplist.hcl
/opt/WTK2.5.2/bin/emulator -classpath ShopList.jar Hecl
```

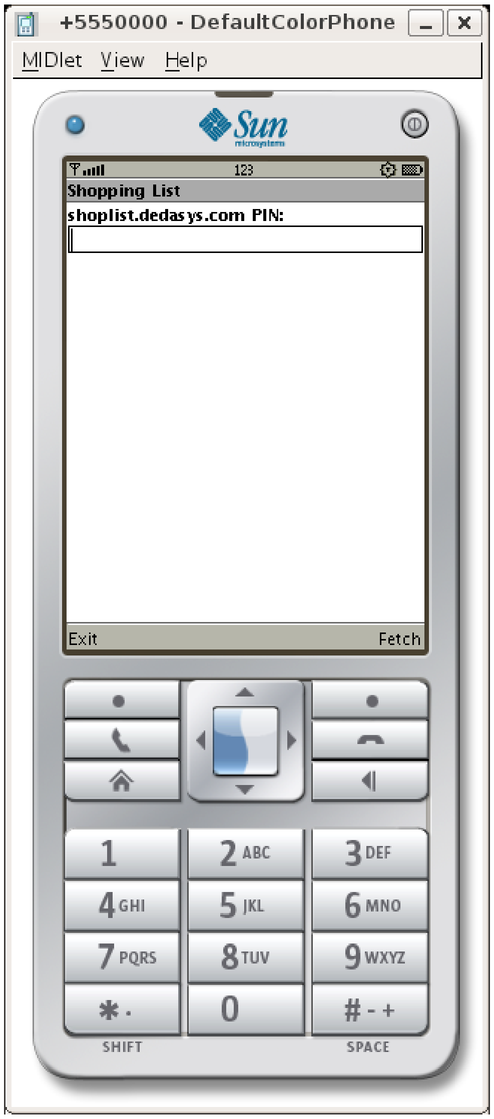
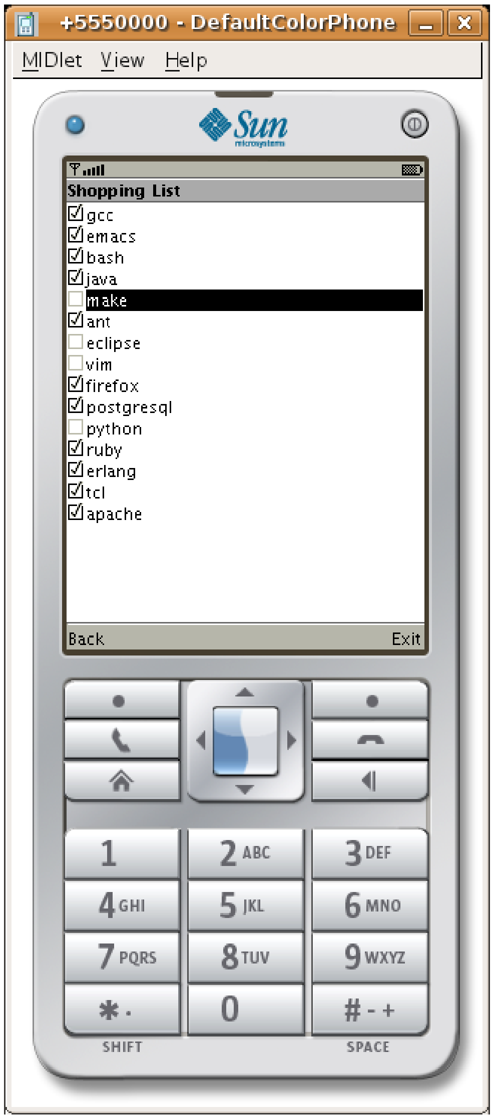


Figure 3: Initial shoplist form

At this point, you enter the PIN number (`346764`), and press the Fetch button. This command executes the code in **MakeList**. The first thing it does is attempt to fetch the data from the shoplist site, using the

http.geturl command. Since this command returns a hash table, in order to get at the data returned, I use the **hget** command to access the "data" element. If the PIN was not available on the server, an error message is returned, and the user is returned to the first screen. Otherwise, a list of checkboxes is created with **lcdui.list**, by specifying "multiple" as the type. Since the shopping list is sent "over the wire" (so to speak...) as a list of lines, all I have to do to add it to the display is split it by lines with the **split** command, and then iterate over that list with **foreach**. The result looks like that displayed figure 4.



Shopping List

- ☑ gcc
- ☑ emacs
- ☑ bash
- ☑ java
- ☐ make
- ☑ ant
- ☐ eclipse
- ☐ vim
- ☑ firefox
- ☑ postgresql
- ☐ python
- ☑ ruby
- ☑ erlang
- ☑ tcl
- ☑ apache

Figure 4: Shopping List

And there you have it, a network-based shopping list in less than 100 lines of code. Of course, there is room for improvement. For instance, in the production version of this shopping list application, RecordStore (in Hecl, the **rms.*** commands make this functionality available) is utilized to save the list and its state between invocations of the program, so that you can leave the application, run it again, and find the list as you left it. Support for multiple lists might also be handy.

Of course, this tutorial barely scratches the surface. Hecl has a number of other GUI commands, and is a complete programming language that can do some interesting and dynamic things. If you're curious, the best way to learn more is to keep reading the documentation, and sign up for the Hecl Google Group [http://groups.google.com/group/hecl], which can be accessed either as a web forum or as a mailing list.

Quick start to developing Java ME apps

Creating new Hecl applications is quick and easy:

1. Create a Hecl script. For starters, you could use the `hello.hcl` script from the tutorial.

2. Use **JarHack** to create a new jar and accompanying jad file:

   ```
   java -jar jars/JarHack.jar -hecljar jars/cldc1.1-midp2.0/Hecl.jar -destdir /tmp
   ```

   This creates `Hello.jar` and `Hello.jad` in the `/tmp` directory, using the `hello.hcl` script. It utilizes the MIDP 2.0 version of Hecl in `jars/cldc1.1-midp2.0/Hecl.jar`

3. To run this code in the emulator:

   ```
   /opt/WTK2.5.2/bin/emulator -classpath /tmp/Hello.jar Hecl
   ```

4. Once you've iterated through a few versions of the script, and are satisfied, you're ready to move it to your phone. Unfortunately, we can't do that for you, as there is no standard method - each phone manufacturer provides their own way. On Linux, I use the following command:

   ```
   gammu nothing --nokiaaddfile Application Hecl
   ```

   Another way to accomplish this is to put the jar and jad files on a web site and download them with your phone's browser.

Hecl J2ME MIDP1.0 Commands

Note

Hecl has different GUI commands for the MIDP1.0 (older phones) and MIDP2.0 (newer). We are in the process of documenting the MIDP2.0 commands.

Commands available in the J2ME MIDP1.0 version of Hecl to interact with the phone. Look in the `midp10/examples` directory for examples of these commands and widgets in use.

Name

alert — Creates an alert

Synopsis

`alert` [label *`title`*] [text *`text`*] [type [{alarm} | {confirmation} | {error} | {error} | {info} | {warning}]]

Description

The lcdui `Alert` class. You must call **setcurrent** to actually display the alert.

Name

choicegroup — Displays a choicegroup in the current form

Synopsis

`choicegroup` [label *`label`*] [type [{exclusive} | {implicit} | {multiple}]] [list *`list`*]

Description

The lcdui `ChoiceGroup` class.

Name

cmd — Adds a command to a form/listbox/textbox

Synopsis

`cmd` [label *`label`*] [type [{back} | {cancel} | {exit} | {help} | {item} | {ok} | {screen} | {stop}]] [code *`code`*]

Description

This is used to create and associate commands with a screen widget (form, listbox, textbox). The lcdui `Command` class.

Name

datefield — Displays an datefield in the current form

Synopsis

`datefield` [label] [typ [{date_time} | {date} | {time}]]

Description

The lcdui `DateField` class.

Name

exit — Exits the application

Synopsis

```
exit
```

Description

Exits the application.

Name

form — Creates a form

Synopsis

`form` [label *`title`*] [code *`code`*]

Description

The lcdui `Form` class. Note that in order to actually display the newly created form, you must call the **setcurrent** command with a reference to the form as an argument.

Name

gauge — Displays an gauge in the current form

Synopsis

`gauge` [label *`label`*] [maxval *`maximum_value`*] [val *`initial_value`*] [interactive [{1} | {0}]]

Description

The lcdui `Gauge` class.

Name

getindex — Fetches a reference to the N'th element in a given form/listbox

Synopsis

`getindex` {*`widget`*} {*`index`*}

Description

Fetches the N'th element in a form.

Name

getprop — Fetches the value of a given property from a widget

Synopsis

`getprop` {*`widget`*} {*`property`*}

Description

Fetches the value of a given property. For example:

```
set tf [textfield label "Insert text:"]
...
set inserted_text [getprop $tf text]
```

In this example, **getprop** fetches the text that has been inserted in the textfield. *`property`* must be a valid property for the given widget.

Name

listbox — Creates a listbox

Synopsis

`listbox` [label *`title`*] [type [{exclusive} | {implicit} | {multiple}]] [code *`code`*] [callback *`code`*]

Description

Similar to the lcdui `ListBox` class, but implemented as a class that extends `Form`, in order to enable the use of callbacks. Like forms and textboxes, it is necessary to use the **setcurrent** command to actually display a listbox.

Name

noscreen — Runs without a current screen widget

Synopsis

`noscreen` {code}

Description

Normally, there is almost always a default screen widget present, so that when items are created, they are automatically added to the screen widget that is in effect. The **noscreen** command executes the code passed to it without a default screen widget.

Name

screenappend — Appends an item to a form or listbox

Synopsis

`screenappend` {*`screen_widget`*} {*`widget`*}

Description

Appends an *`widget`* to the form or listbox *`screen_widget`*.

Name

setcurrent — Displays an alert/form/listbox/textbox

Synopsis

`setcurrent` {screen_widget}

Description

Screen widgets (alerts, forms, listboxes and textboxes), are not displayed when created. **setcurrent** displays them. For example:

```
set f [form label "New Form"]
setcurrent $f
```

Name

setindex — Sets the *`index`*th element in a form/listbox to specified item

Synopsis

`setindex` {*`widget`*} {*`index`*} {*`newitem`*}

Description

Sets the *`index`*'th item of a form/listbox to item *`newitem`*.

Name

setprop — Sets a given property of a widget

Synopsis

`setprop` {*`widget`*} {*`property`*} {*`new_value`*}

Description

Sets the value of a given widget's property.

Name

string — Adds a string to the current form.

Synopsis

`string` {*`string`*}

Description

Appends the specified *`string`* to the current form.

Name

stringitem — Displays a stringitem in the current form

Synopsis

`stringitem` [label *`label`*] [text *`text`*]

Description

The lcdui `StringItem` class.

Name

textbox — Creates a textbox

Synopsis

`textbox` [label *`title`*] [len *`length_in_characters`*] [type [{any} | {emailaddr} | {numeric} | {phonenumber} | {passwd} | {url}]] [text *`text`*] [code *`code`*]

Description

The lcdui `TextBox` class. To display a textbox, you must use the **setcurrent** command.

Name

textfield — Displays a textfield in the current form

Synopsis

`textfield` [label *`label`*] [len *`length_in_characters`*] [type [{any} | {emailaddr} | {numeric} | {phonenumber} | {passwd} | {url}]] [text *`text`*]

Description

The lcdui `TextField` class.

Hecl Java ME MIDP2.0 Commands

The MIDP 2.0 commands in Hecl are more complete and more advanced than the 1.0 commands, and should be used if possible. This is a fairly complete list of which devices are MIDP2.0 capable: http://devices.j2mepolish.org/interactivedb/searchdevices.faces

To fully understand these commands, a perusal of the documentation here is likely a good idea: http://java.sun.com/javame/reference/apis/jsr118/ In particular, look at the lcdui package.

This code was originally written and contributed to Hecl by Wolfgang Kechel. Thanks!

Name

lcdui.alert — Pops up an alert

Synopsis

`lcdui.alert` [-title *`title`*] [-ticker *`tickerwidget`*] [-commandaction *`commandActionProc`*] [-type { {info} | {warning} | {error} | {alarm} | {confirmation} }] [-text *`text`*] [-timeout { {*`milliseconds`*} | {forever} }] [-indicator { {1} | {0} }]

`$alertcmd` {setcurrent} [cget *`-optname`*] [configure *`-optname`* *`optval`*]

Description

The **lcdui.alert** command creates an alert to let the user know that something unusual has happened, or to ask for confirmation. The object returned from **lcdui.alert** is itself a command, that can be called with a subcommand `setcurrent` to make the alert the currently displayed item. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.Alert [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Alert.html]

Example

```
[lcdui.alert -title "Reactor core meltdown" -text \
    "You went and pressed the red button, didn't you!" \
    -timeout forever] setcurrent
```

Live example: http://www.heclbuilder.com/scripts/show/141

Name

lcdui.canvas — Creates a canvas

Synopsis

`lcdui.canvas` [-autoflush { {1} | {0} }] [-cmdbg *`color`*] [-cmdfg *`color`*] [-eventhandler *`code`*] [-fullscreen { {1} | {0} }] [-title *`title`*] [-suppresskeys { {1} | {0} }]

`$canvascmd` [cget *`-optname`*] [configure *`-optname`* *`optval`*] [flush [*`x y width height`*]] [graphics] [repaint] [servicerepaints]

Description

The **lcdui.canvas** command creates a canvas that can be used to draw arbitrary items, such as rectangles, text, circles, and respond to keyboard events. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.game.GameCanvas [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/game/GameCanvas.html]

The options describing the canvas are as follows:

- `-autoflush`: Takes a boolean argument (1 or 0) indicating FIXME
- `-cmdbg`: Selects a background color for the command background.
- `-cmdfg`: Selects a background color for the command foreground.
- `-eventhandler`: The Hecl code to execute when an event is generated.
- `-fullscreen`: A boolean switch specifying whether to run in "full screen" mode. "Full screen" mode means that commands, which would normally be accessible via certain keys on the cell phone, are not accessible, and that those keys generate regular events instead of calling commands.
- `-suppresskeys`: A boolean switch specifying whether to suppress key events or not. Suppressing these events, if the application does not need to keep track of them, may improve performance some.

The **lcdui.canvas** command returns an object that is itself a command, and can be called with several subcommands:

- **flush**: With no argument, flushes (displays) the off-screen buffer to the visible display. With arguments *`x y width height`*, flushes the specified section to the screen.
- **graphics**: Returns a graphics object that acts as a command to manipulate the Canvas' graphics buffer.
- **repaint**: Repaints the entire canvas.
- **servicerepaints**: Forces any repaints that haven't occured yet to be performed immediately.

Example

```
set canvas [lcdui.canvas -title "Test Canvas" -commandaction backToMainMenu \
    -eventhandler canvasEvents]
set graphics [$canvas graphics]
$graphics frect [list 10 10] [list 10 80]
$graphics frect [list 80 10] [list 10 80]
```

```
$graphics frect [list 10 50] [list 80 10]
$canvas setcurrent
```

Live example: http://www.heclbuilder.com/scripts/show/142

Name

lcdui.choicegroup — Creates a group of potential choices that can be added to a form.

Synopsis

`lcdui.choicegroup` [-label *`label`*] [-type { {exclusive} | {multiple} | {popup} }] [-fit { {default} | {on} | {off} }]

`$choicegroupcmd` [cget *`-optname`*] [configure *`-optname`* *`optval`*] [append *`text`* [*`image`*]] [delete *`itemnum`*] [deleteall] [insert *`position`* *`text`* [*`image`*]] [itemcget *`itemnum`* *`-option`*] [itemconfigure *`itemnum`* *`-option`* *`value`*] [selection { {clear [*`index`*] } | {index} | {get} | {gettext} | {set *`index`*} }] [size]

Note

Choicegroups also utilize the common "item" options, described here: lcdui item options

Description

The **lcdui.choicegroup** command creates a "ChoiceGroup" widget that can be added to a form. A ChoiceGroup is a list of options that can be selected. ChoiceGroups come in three forms: radio buttons (the `exclusive` type), where only one at a time may be selected, check boxes (the `multiple` type), or as a popup. (check boxes). For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.ChoiceGroup [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/ChoiceGroup.html]

The options describing the choicegroup are as follows:

- `-label`: Labels the choicegroup with the supplied text.
- `-type`: Determines the type of the chiocegroup, which can be one of `multiple` (check boxes), `exclusive` (radio buttons), or a `popup`
- `-fit`: Specifies the type of word-wrapping desired. `default` means that the application has no preferences and that the default can be used. The `on` option means that text wrapping should be performed to fit long lines. The `off` option means that text wrapping should not be performed and that long lines will be truncated, and there should be some indication of this truncation such as an ellipsis.

The **lcdui.choicegroup** command returns an object that is itself a command, and can be called with several subcommands:

- **append**: Appends the given text to the choicegroup as one of the possible selections. Optionally, an image (as generated by **lcdui.image** may also be specified.
- **delete**: Deletes the item specified by *`itemnum`*
- **deleteall**: Deletes all items in the given choicegroup.
- **insert**: Inserts the given text, and optional image at *`position`*
- **itemcget**: Given an item and an *`-option`*, which can be one of `-font`, `-text`, `-image`, `-selected`, returns the requested information, with `-selected` returning 1 if the item is selected, 0 if it isn't.
- **itemconfigure**: Instead of returning information about the options listed above, sets the specified option for the given item.

- **selection**: Used with a subcommand of its own, which can be one of:
  - **get**: Returns a list of integers indicating which elements are selected.
  - **gettext**: Returns the text of the selected element.
  - **set**: Selects the specified item.
  - **clear**: Clears the selection from the given item, or, if no *`itemnum`* is given, clears all selected items.
- **size**: Returns the number of elements in the choicegroup.

Example

```
set choicegroup [lcdui.choicegroup -label "60ies bands"]
$choicegroup append "Beatles"
$choicegroup append "Beach Boys"
$choicegroup append "Rolling Stones"
set form [lcdui.form -title "ChoiceGroup example form"]
$form append "Pick one:"
$form append $choicegroup
$form setcurrent
```

Live example: http://www.heclbuilder.com/scripts/show/143

Name

lcdui.command — Creates a command that can be attached to a screen

Synopsis

`lcdui.command` [-label *`label`*] [-longlabel *`label`*] [-priority *`priority`*] [-type { {back} | {cancel} | {exit} | {help} | {item} | {ok} | {screen} | {stop} }]

Description

The **lcdui.command** command creates a command that can be attached to a Screen or an Item. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.Command [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Command.html]

The options describing the choicegroup are as follows:

- `-label`: A short label for the command, which should be as short as possible.
- `-longlabel`: A long, more descriptive label.
- `-priority`: A positive number indicating the order of appearance of commands, 1 being the highest priority.
- `-type`: Indicates the type of command. The command types are documented here [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Command.html#field_summary]

Example

```
set exitcmd [lcdui.command -label Exit -longlabel Exit \
    -type exit -priority 2]
set backcmd [lcdui.command -label Back -longlabel Back \
    -type back -priority 1]
set addtextcmd [lcdui.command -label AddText -longlabel AddText \
    -type screen -priority 1]

proc NewForm {} {
    global exitcmd
    global backcmd
    global addtextcmd
    set form [lcdui.form -title "Commands" -commandaction HandleCmd]
    $form setcurrent
    $form addcommand $exitcmd
    $form addcommand $backcmd
    $form addcommand $addtextcmd
}

proc HandleCmd {cmd form} {
   global exitcmd
   global backcmd
   if { eq $cmd $backcmd } {
    NewForm
   } elseif { eq $cmd $exitcmd } {
    [lcdui.alert -title "Goodbye" -text "Goodbye!" -type info \
            -timeout forever] setcurrent
        after 1000 midlet.exit
   }
   $form append "Blah blah"
}

NewForm
```

Live example: http://www.heclbuilder.com/scripts/show/144

Name

lcdui.date — Date/Time widget for lcdui forms

Synopsis

`lcdui.date` [-date *`date/time in seconds`*] [-label *`label`*] [-type { {date} | {date_time} | {time} }]

`$datecmd` [cget *`-optname`*] [configure *`-optname`* *`optval`*]

Note

Date fields also utilize the common "item" options, described here: lcdui item options

Description

The **lcdui.date** command creates a widget that can either display, or allow the user to enter a date, a time, or both. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.DateField [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/DateField.html]

The **lcdui.date** command takes these options:

- `-label`: A label for the date widget.
- `-type`: One of `date`, which only displays a date selection widget, `time`, which only displays a time selection widget, or `date_time`, which allows the user to select both a date and a time.
- `-date`: The current time, expressed in milliseconds.

Example

```
set date [lcdui.date -label "Current date and time:" \
    -date [clock time] -type date_time]
set form [lcdui.form -title "Date Form"]
$form append $date
$form setcurrent
```

Live example: http://www.heclbuilder.com/scripts/show/144

Name

$eventcmd — Command/object describing a Canvas event.

Synopsis

$eventcmd [cget *-optname*] [configure *-optname* *optval*]

Description

The ***$eventcmd*** command is used to access information about events received by a Canvas. There is no Hecl command to create event commands - they are passed in as arguments to the `-eventhandler` specified for an **lcdui.canvas**.

There are a number of parameters that can be queried:

- `-canvas`: Returns the canvas command associated with the event's canvas.
- `-reason`: A number representing one of the following reasons:

```
public static final int E_NONE = 0;
public static final int E_PAINT = 1;
public static final int E_PPRESS = 2;
public static final int E_PRELEASE = 3;
public static final int E_PDRAG = 4;
public static final int E_KPRESS = 5;
public static final int E_KRELEASE = 6;
public static final int E_KREPEAT = 7;
public static final int E_HIDE = 8;
public static final int E_SHOW = 9;
public static final int E_RESIZE = 10;
public static final int E_UNKNOWN = -1;
```

  Which correspond to:

  - `E_PAINT`: Used when the Canvas' `paint` method is called.
  - `E_PRESS`: Used when the Canvas' `pointerPressed` method is called.
  - `E_PRELEASE`: Used when the Canvas' `pointerReleased` method is called.
  - `E_PDRAG`: Used when the Canvas' `pointerDragged` method is called.
  - `E_KPRESS`: Used when the Canvas' `keyPressed` method is called.
  - `E_KRELEASE`: Used when the Canvas' `keyReleased` method is called.
  - `E_KREPEAT`: Used when the Canvas' `keyRepeated` method is called.
  - `E_HIDE`: Used when the Canvas' `hideNotify` method is called.
  - `E_SHOW`: Used when the Canvas' `showNotify` method is called.
  - `E_RESIZE`: Used when the Canvas' `sizeChanged` method is called.

  - `E_UNKNOWN`: Used to signify an unknown event.

- `-x`: The x coordinate of the event.
- `-y`: The y coordinate of the event.
- `-width`: The width of the screen area covered by the event.
- `-height`: The height of the screen area covered by the event.
- `-keycode`: An integer referencing the key involved in the event.

  Note

  Be aware that different phones may use different numbers.

- `-keyname`: Returns a string giving a cross platform name for the key pressed. Names are given here: http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Canvas.html#field_summary, with the addition of "LEFT_SK" and "RIGHT_SK" for the left and right soft keys These only occur when in full screen mode and the soft key events go to the program)
- `-gameaction`: Returns the integer code for the game action. Described in further detail here: http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Canvas.html#gameactions

Example

See the example script in the midp20/ directory. http://github.com/davidw/hecl/blob/master/midp20/script.hcl

Name

lcdui.font — Font information and manipulation command.

Synopsis

`lcdui.font` [names] [*`fontname`* | {cget} { {-face} | {-size} | {-plain} | {-bold} | {-italic} | {-underlined} | {-height} | {-baselineposition} }] [*`fontname`* charwidth *`string`* [*`offset`* [*`length`*]]] [*`fontname`* stringwidth *`string`* [*`offset`* [*`length`*]]]

Description

The **lcdui.font** command is used to fetch information about fonts. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.Font [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Font.html]

It accepts several subcommands:

- **names**: Returns a list of all the available fonts.
- *`fontname`* **charwidth** *`string`*: Returns the width, in pixels, of the given characters. Takes optional *`offset`* and *`length`* parameters.
- *`fontname`* **stringwidth** *`string`*: Returns the width, in pixels, of the given string. Takes optional *`offset`* and *`length`* parameters.
- *`fontname`* **cget** *`-option`*: Used to fetch information about a font. Allowed options are as follows:
  - `-face`: One of "system", "proportional", or "monospace".
  - `-size`: One of "small", "medium" or "large".
  - `-plain`: Returns a 1 if the font is "plain" - not bold, underlined or italic, or 0 if it isn't.
  - `-bold`: Returns 1 or 0 if the font is bold or not.
  - `-italic`: Returns 1 or 0 if the font is italic or not.
  - `-underlined`: Returns 1 or 0 if the font is underlined or not.
  - `-height`: Returns the standard height of a line of text in this font, in pixels.
  - `-baselineposition`: Returns the distance in pixels from the top of the text to the text's baseline.

Example

```
set form [lcdui.form -title "Font examples"]

$form setcurrent

foreach f [sort [lcdui.font names]] {
    set s [lcdui.stringitem -text $f]
    $s configure -font $f
    $form append $s
}
```

Live example: http://www.heclbuilder.com/scripts/show/147

Name

lcdui.form — Creates a form that can contain various widgets

Synopsis

`lcdui.form` [-title *`title`*] [-ticker *`tickerwidget`*] [-commandaction *`commandActionProc`*]

`$formcmd` [cget *`-optname`*] [configure *`-optname optval`*] [size] [append { {*`string`*} | {*`image`*} | {*`widget`*} }] [item *`itemnum`*] [delete *`itemnum`*] [deleteall] [setcurrent]

Description

The **lcdui.form** command creates a form, returning a command/object that can be used to further manipulate the created form. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.Form [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Form.html]

The subcommands accepted by the ***`$formcmd`*** that **lcdui.form** returns are as follows:

- **cget**: Fetch configuration options.
- **configure**: Set configuration options.
- **item** *`itemnum`*: Return the item identified by *`itemnum`*.
- **delete** *`itemnum`*: Delete the item identified by *`itemnum`*.
- **deleteall**: Delete all items associated with this form.

Example

```
set choicegroup [lcdui.choicegroup -label Choice -type popup]
foreach x {c1 c2 c3} {
    $choicegroup append $x
}
set ticker [lcdui.ticker -text "I am a Ticker!"]

set form [lcdui.form -title "Example Form"]
$form setcurrent
$form configure -ticker $ticker
$form append [lcdui.textfield -label "TextField" -text "TextField" -uneditable 1]
$form append [lcdui.textfield -label "Editable TextField" -text "editable text"]
$form append [lcdui.imageitem -image $logo -anchor center]
$form append [lcdui.spacer -label spacer1 -minwidth 200 -minheight 2]
$form append [lcdui.stringitem -text "Stringitem"]
$form append [lcdui.spacer -label spacer2 -minwidth 200 -minheight 4]
$form append [lcdui.date -label "Date/Time" -date [clock time]]
$form append $choicegroup
$form append [lcdui.imageitem -image $logo]
$form append [lcdui.gauge -label "How cool is Hecl?" -interactive 1 \
       -value 10 -maxvalue 10]
```

Live example: http://www.heclbuilder.com/scripts/show/148

Name

lcdui.gauge — Creates a gauge widget that can be attached to a form.

Synopsis

`lcdui.gauge` [-label *`label`*] [-interactive { {1} | {0} }] [-value *`value`*] [-maxvalue *`maxvalue`*]

`$gaugecmd` [cget *`-optname`*] [configure *`-optname`* *`optval`*]

Note

Gauge items also utilize the common "item" options, described here: lcdui item options

Description

The **lcdui.gauge** command creates a gauge element, and returns a command/object that can be used to manipulate it. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.Gauge [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Gauge.html]

The options for this command are as follows:

- `-value`: An integer giving the initial value of the gauge, or, one of these special values: "continuous-idle", "continuous-running", "incremental-idle", or "incremental-updating". These values are utilized for non-interactive guages that can be used as progress meters. These values are explained in further detail on the javadoc page linked above.

  `-maxvalue`: An integer giving the maximum value of the guage, or "indefinite" if there is no maximum value.

  `-interactive`: Either 1 or 0, depending on whether the gauge is interactive, or is for display purposes only.

Example

```
set form [lcdui.form -title "Gauge Example"]
$form append [lcdui.gauge -label "From 1 to 10" -interactive 1 \
       -value 6 -maxvalue 10]
$form setcurrent
```

Live example: http://www.heclbuilder.com/scripts/show/149

Name

lcdui.image — Creates an image.

Synopsis

`lcdui.image` [-file *`filename`*] [-rms *`recordstore`*] [-resource *`resourcename`*] [-image *`image`*] [-data *`data`*]

`$imagecmd` [cget *`-optname`*] [configure *`-optname optval`*] [thumbnail [thumbwidth [thumbwidth]]] [graphics]

Description

The **lcdui.image** command creates an image, from some data, either by indicating a filename, record store, or a resource name. It returns an object/command that can be utilized to manipulate the image. Mutable images are those loaded from existing data, generally. Immutable images are created as a "blank slate", with white pixels. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.Image [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Image.html]

The options available for the **lcdui.image** command are described below:

- `-file`: Load the image from the given file name.
- `-rms`: Load the image from the given record store.
- `-resource`: Load the image from the given resource. By "resource", we mean a file included in the .jar.
- `-image`: Create a new image from an existing image.
- `-data`: Create a new image from binary data.

The subcommands available to an ***`$imagecmd`*** are listed below:

- **thumbnail**: Creates a thumbnail from the image.
- **graphics**: Unless the image is immutable, creates and returns a new graphics object associated with the image.

Creating base64 encoded images

There are two ways to include images in Hecl applications. One is to add the image to the .jar file itself. The second is to include it directly in the script, as a base64 included string. Here's how to do so on a Linux machine:

1. Create an image. For example, with "The Gimp", an image manipulation program on Linux, you might create a PNG, `x.png`.
2. Now you need to turn the binary PNG file into a string that can be included in a script:

   ```
   base64 < x.png > x.txt
   ```

3. Now copy and paste x.txt into your script, like in the script below

Example

```
# base64 encoded PNG
set data {iVBORw0KGgoAAAANSUhEUgAAAGQAAABkCAYAAABw4pVUAAAAAXNSR0IArs4c6QAAAAZiS0dE
/wD/oL2nkwAAAAlwSFlzAAALEwAACxMBAJqcGAAAAAd0SU1FB9gKGBQIB6A2RzQAAAAZdEVYdENv
bW1lbnQAQ3JlYXRlZCB3aXRoIEdJTVBXgQ4XAAAG40lEQVR42u2dW0hUXRTH98w4Upk145gylRb1
YOhMNyIigkC65/2WYCARhgS9RXTDCHqyIiF66e7dHiywIBB66cHoQhko9DAP0zTpjIo4WJTp+O/J
j+/rM/caZ85tXH84b+fsvdb6edaZfc5aWxMACJZuZOYQMBAWA2EgLAbCQFgMhIGwGAgDYTEQFgNh
ICwGwkBYDISBsBgIA2EtKCDl5eXCZDLNeaxfv158//5ds0B8+/ZNrFu3TmpnRUWFsoZABQ0NDSE1
NRVCiDmPkydPQivV1tZK7VuxYgWGhoYUtUOo5XBHR4fUYZPJhBcvXqgOo7u7W2qbEAKPHj1S3Bah
puMlJSVSp9euXYvx8XHVbAqFQsjIyJDaVVZWpoo9qgIJBoNwOBxS50+cOKGaTceOHZPak5qaqaniq
0gQIALS1tZHSQ3d3t+K2PHv2jGRLR0eHavFRHQgAFBUVSYOQkZGBUCikmA2jo6NwOp1SO0pLS1WN
jSZAAoEAUlJSpME4fvy4YjZUVVWRUlUwGIx/IADQ0tJCShfPnz+P+dyPHz8mzd3e3q56XDQDAgAF
BQXSoKxatQpjY2Mxm3N4eBhpaWnSeUtKSjSJiaZABgcHYbfbpcGprq6O2ZxlZWXS+RwOh+qpShdA
AKCpqYmUPp4+fRr1XO3t7aS52traNIuH5kAAIC8vTxokp9OJ0dHRqO5Gyg+J4uJiTWOhCyBfv36F
zWaTBquqkrR55XD4UAgEGAgAPDw4UNSOnny5EnEYzc2NpLGbm1t1TwOugECAIcOHZIGLT09HSMj
I+Qx/X4/6e4rKirSRQx0BYQavMrKSvKYBw4ckI6XearSJRAAeSJRAAePDgASm9dHZ2Sse6ffs2aayW
lhbd+K87INTUlZaWhuHh4b+O4fV6kZycLB2nsLBQV77rEojf78fy5ulwSwvL5/1+unpaeTm5pJS
1eDgIAOh6N69e/N+NX7z5k3StPc3NzbrzW7dAqA/kP9/IejweJCUlSa8r8KCjQpc+6BuLz/
dZOC7OxsTExMMBCZGYGSE08ly5d+usY1D6Quro6BiJTfn6+NJCbNm3Cr1+/4NZWGSa8rKC9Q
6jocDmPXrl3S8+12u+5SlSGAAMCdO3fIi7rr16+Tzr1x44ba0fr6u
fTUB+t98xufzCZfLJcbHx6Max263i/7+fuF0OrmUNBplZmaKa9euRT1OQ0ODrmGoVrkYK+3Zs4f0
jJjtyMvLM4SPhkhZM/r8+bNwu90Rpy6bzSb6+/vFypUrudg6llqzZo2or6+fV6oyAgzDpayZ1yI7
d+4kp6rDhw8byj9DpSwhhPD7/cLlcolQKCQ912KxCK/XK1avXs39IUqppqaGBEMIIcLhsOjq6jKW
g0a6nakvHP99JCcnw+fzGcZHHwwChvpI3+nPEMEAOHjw47zWI1rVWcQfk/v37UcGYaUeLpDiCgcyR
qiiFD5Tj6NGjDCRaUb6vb926FZWVlZpV0y8YIJQKFKvVio8fP2JkZATp6enS8zMzM1XtYYwbINTy
0suXL/9zTWdnJ+kuOVEivM5vN6OnpYSBUUep8rVYrent7/3ct
dZOC7OxsTExMMBCZBgYGSE08ly5d+usY1D6Quro6BiJTfn6+NJCbNm3Cr1+/5hyH0umbmJiIvr4+
BvI3UbqprFYrPnz4IB2L2i63Y8cOhMNhBjLfVBVJmqH2hTQ0NDCQP0XpcNq4caM0Vc1nYZmUlASv
18tAZtTc3CwNWkJCAt6/fx/F++fCFVP+7fv5+BRJLrL168O85qNWPjY2NDKSwsFAaKLfbHWa
Ye/evaT2BK3603UBhLK9RkJCAt69exf1XF6vF0uXLpXOd+TIkYUJhNo3fuHChZjjNeevWLVLq6urq
WnhAKAs3l8sV09cb09PT2L17N2l/FSW3htIdkNbWVmlQLBYL3r59G/O5XG56PR4PlixZ
BAKkbf7OnTunmA03btwgbcj58uXL+AdSXFwsDUZOTg7EYGhdiQLbBAIIMHKbIHD4jDZ7fuI
8ObNG8Vt+fTpE6nB5/z58/EdlY+nvSze7bvLoYHQtmzNzs7W9UUMTU1he3bt0vt
2rZtG6ampuIHCOWDkcViwevXrxV7V/iPb19SExMVFq39WrV+MDSDAYJH1SPXPmjGaLsStXrkjtW7x4
MTwej/GBlJaWSp3dsGEDfvz4oRmQyclJbNmyRWpnbm6usYFQNuA3m8149eqV1u840dvbC6vVKrX3
7t27xgRCrf44ffo09KKKigppvKxaK/Kb56ujqpvTabDQMDA8YDQtmONSsrS9NU9acmJibgdrs13dPXcC1t/C+PWAyE
gbAYCIuBMBAWA2EgLAbCQFgMhIGwGAhrFv0GfQwoE8oHj1AAAAAASUVORK5CYII=}

set logo [lcdui.image -resource /hecl_logo.png]
set ximg [lcdui.image -data [base64::decode $data]]
set form [lcdui.form -title "Show Image"]
$form append $logo
$form append $ximg
$form setcurrent
```

Live example: http://www.heclbuilder.com/scripts/show/150

Name

lcdui.imageitem — An item that can contain an image, in order to attach it to a form.

Synopsis

`lcdui.imageitem` [-appearance { {plain} | {button} | {hyperlink} }] [-image *`image`*] [-label *`label`*] [-text *`text`*]

`$imageitemcmd` [cget *`-optname`*] [configure *`-optname`* *`optval`*]

Note

Image items also utilize the common "item" options, described here: lcdui item options

Description

The **lcdui.imageitem** command creates an item holding an image, in order to be able to attach an image to a form. It also returns a command/object that can be used to manipulate the ImageItem. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.ImageItem [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/ImageItem.html]

The options unique to this command are as follows:

- `-appearance`: Specifies the appearance and behavior of the imageitem. The meaning of the various values is explained in further detail here: javax.microedition.lcdui.Item [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Item.html#appearance].

Name

lcdui.list — Creates a full-screen list

Synopsis

`lcdui.list` [-title *`title`*] [-type { {exclusive} | {multiple} | {implicit} }] [-selectcommand *`cmd`*]

Description

The **lcdui.list** command creates a full-screen list and returns a command/object that can be used to manipulate it. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.List [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/List.html]

The options unique to this command are as follows:

- `-type`: One of `exclusive`, `multiple` or `implicit`. Exclusive and multiple correspond, respectively, to radio buttons and checkboxes. Implicit lists act like a menu, dispatching to the list's `-commandaction` when a list item is selected.
- `-selectcommand`: Select the command which should be used for implicit list select actions.

Example

```
set selectcmd [lcdui.command -label Select -longlabel Select -type \
    item -priority 1]
set lst [lcdui.list -title "List Example" -commandaction selectname]
set names {Anna Barbara Carla Daniela Emanuela Federica}
foreach n $names {
    $lst append $n
}
$lst setcurrent
$lst addcommand $selectcmd
proc selectname {cmd lst} {
    global names
    [lcdui.alert -text "Selected: [lindex $names [$lst selection get]]"] \
        setcurrent
}
```

Produces:

Live example: http://www.heclbuilder.com/scripts/show/151

Name

lcdui.settings — Returns or sets information about the graphical environment.

Synopsis

`lcdui.settings` [-color] [-alphalevels] [-alertimagewidth] [-alertimageheight] [-listimagewidth] [-listimageheight] [-choiceimagewidth] [-choiceimageheight] [-bg] [-fg] [-hilightbg] [-hilightfg] [-border] [-hilightborder] [-borderstyle] [-hilightborderstyle] [-cvfullscreen [{1} | {0}]] [-cvdocmds [{1} | {0}]] [-cvkeepcmdsinfullscreen [{1} | {0}]] [-cvcmdbg [color]] [-cvcmdfg [color]] [-skleft [keycode]] [-skright [keycode]]

Description

The **lcdui.settings** command returns information about the phone the program is running on.

- `-color`: is this a color display? 1 or 0
- `-alphalevels`: number of alpha levels
- `-alertimagewidth`: width in pixels of the alert image
- `-alertimageheight`: height in pixels of the alert image
- `-listimagewidth`: width in pixels of the list image
- `-listimageheight`: height in pixels of the list image
- `-choiceimagewidth`: width in pixels of the choice image
- `-choiceimageheight`: height in pixels of the choice image
- `-bg`: background color
- `-fg`: foreground color
- `-hilightbg`: background hilight color
- `-hilightfg`: foreground hilight color
- `-border`: border color
- `-hilightborder`: border hilight color
- `-borderstyle`: border style - can be either DOTTED (1) or SOLID (0)
- `-hilightborderstyle`: border highlight style - can be either DOTTED (1) or SOLID (0)
- `-cvfullscreen`: Get/set whether full screen mode is allowed for the canvas widget.

  This setting enables the use of fullscreen mode for a canvas. The lcdui `Canvas` class behaves different in fullscreen mode: it does not show commands. Full screen mode is enabled by default, so a canvas (being a subclass of `lcdui.game.GameCanvas`) can be created in fullscreen mode. On some devices the behavior in fullscreen mode is broken, so this setting gives you the chance to avoid the fullscreen usage without changing the application itself but only the application startup that might detect certain phone features and turn off fullscreen mode generally for all canvases.

Devices being picky in fullscreen mode include most of the BlackBerry devices since handling of commands is totally different from other Java ME devices. Most BlackBerry devices use the dial to select between command entries.

- `-cvdocmds`: Get/set whether to display commands in the full screen canvas or not.

  Enables the display of commands by the hecl canvas. When it fullscreen mode, canvases do not show commands. You may want a canvas in fullscreen mode and wish to use commands as usual, so the canvas in hecl provides a fallback mechanism to show commands attached to the canvas by itself.

- `-cvkeepcmdsinfullscreen`: Get/set whether to add commands in full screen mode.

  Enables treatment of commands attached to a canvas even in fullscreen mode. This behavior is implemented in the Hecl midp layer and may be slidely different from the regular phone behavior for commands attached to other lcdui interafce elements like a form or list.The setting has an effect only for a canvas in fullscreen mode and enables command handling similar to the command handling of non-fullscreen canvases.

- `-cvcmdbg`: Get/set color value for command background color in canvas, when Hecl's full screen Canvas command handling is enabled (see above).

- `-cvcmdfg`: Get/set color value for command foreground color in canvas, when Hecl's full screen Canvas command handling is enabled (see above).

- `-skleft`: Get/set the numeric keycode for the left soft key.

  The values are used to identify the keys triggering command handling for a canvas. The standard settings for these values are the usual keycode -6 (left) and -7 (right) that work on most Java ME devices. There are some well-known differences that are detected by the Hecl midp package:

  Siemens S65 and friends: -1 (left), -4 (right) - detected via existence of class `com.siemens.mp.lcdui.Image` BlackBerry: 524288=0x80000 (left), 0x1b0000 (right) - detected via existence of class `net.rim.device.api.system.Application`

  Application developers can tailor these settings for specific device models in their own applications.

- `-skright`: Get/set the numeric keycode for the right soft key (see explanation above).

Example

```
set form [lcdui.form -title "Settings Demo" -commandaction menu1sel]
$form setcurrent

foreach s {"-color"
    "-alphalevels"
    "-alertimagewidth"
    "-alertimageheight"
    "-listimagewidth"
    "-listimageheight"
    "-choiceimagewidth"
    "-choiceimageheight"
    "-bg"
    "-fg"
    "-hilightbg"
    "-hilightfg"
    "-border"
    "-hilightborder"
    "-borderstyle"
    "-hilightborderstyle"
} {
    $form append [lcdui.stringitem -label "[strtrim $s -]:" -text \
        [lcdui.settings cget $s]]
}
```

Live example: http://www.heclbuilder.com/scripts/show/152

Name

lcdui.spacer — Creates a spacer element in a form.

Synopsis

`lcdui.spacer` {-minwidth *`minimum width`*} {-minheight *`minimum height`*}

Note

Image items also utilize the common "item" options, described here: lcdui item options

Description

The **lcdui.spacer** command creates a spacer to separate form elements. Via the `-minwidth` and `-minheight` options, you can specify the minimum width and height. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.Spacer [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Spacer.html]

Name

lcdui.stringitem — Creates a string item that can be added to a form.

Synopsis

`lcdui.stringitem` {-label *`label`*} {-text *`text`*}

`$stringitemcmd` [cget { {-appearance} | {-text} | {-font} }] [configure { {-appearance { {plain} | {button} | {hyperlink} } } | {-text *`text`*} | {-font *`font`*} }]

Note

String items also utilize the common "item" options, described here: lcdui item options

Description

The **lcdui.stringitem** command creates a string item (label + text) that can be attached to a form. Normally, the label is bold, and the text is in the default font. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.StringItem [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/StringItem.html]

Name

lcdui.textbox — Creates a text box for editing larger blocks of text.

Synopsis

`lcdui.textbox` {-title *`title`*} {-text *`text`*} {-maxlen *`maxlen`*} {-type { {any} | {emailaddr} | {numeric} | {phonenumber} | {decimal} } }

`$textboxcmd` [cget { {-type} | {-text} | {-maxlen} | {-password} | {-uneditable} | {-sensitive} | {-non_predictive} | {-initial_caps_word} | {-caretposition} }] [configure { {-type *`type`*} | {-text *`text`*} | {-maxlen *`maxlen`*} | {-password { {1} | {0} } } | {-uneditable { {1} | {0} } } | {-sensitive { {1} | {0} } } | {-non_predictive { {1} | {0} } } | {-initial_caps_word { {1} | {0} } } }]

Description

The **lcdui.textbox** command creates a full-screen text editing widget that can be used for editing longer, multi-line chunks of text. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.TextBox [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/TextBox.html]

There are several options governing the functionality of this widget. Note that more than one constraint may be selected at a time:

- `-password`: If 1, obscure the data as it is entered.
- `-uneditable`: If 1, the user is not allowed to edit the displayed text.
- `-sensitive`: If 1, "indicates that the text entered is sensitive data that the implementation must never store into a dictionary or table for use in predictive, auto-completing, or other accelerated input schemes."
- `-non_predictive`: Don't use the local text-completion system if this option is set.
- `-initial_caps_word`: The initial letter of each word should be capitalized if this option is set.
- `-initial_caps_sentence`: The initial letter of each sentence should be capitalized if this option is set.

Example

```
set evalcmd [lcdui.command -label Evaluate -longlabel Evaluate]
set textbox [lcdui.textbox -text {[lcdui.alert -text "hi!"] setcurrent} \
 -commandaction runCode]
$textbox setcurrent
$textbox addcommand $evalcmd

proc runCode {cmd txtbox} {
    eval [$txtbox cget -text]
}
```

Live example: http://www.heclbuilder.com/scripts/show/153

Name

lcdui.textfield — Creates a small text editing widget that can be attached to a form.

Synopsis

`lcdui.textfield` {-label *`label`*} {-text *`text`*} {-maxlen *`maxlen`*} {-type { {any} | {emailaddr} | {numeric} | {phonenumber} | {decimal} } }

`$textfieldcmd` [cget { {-type} | {-text} | {-maxlen} | {-password} | {-uneditable} | {-sensitive} | {-non_predictive} | {-initial_caps_word} | {-caretposition} }] [configure { {-type *`type`*} | {-text *`text`*} | {-maxlen *`maxlen`*} | {-password { {1} | {0} } } | {-uneditable { {1} | {0} } } | {-sensitive { {1} | {0} } } | {-non_predictive { {1} | {0} } } | {-initial_caps_word { {1} | {0} } } }]

Description

The **lcdui.textfield** command creates a one-line text editing widget that can be attached to a form. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.TextBox [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/TextBox.html]

Example

```
set evalcmd [lcdui.command -label Evaluate -longlabel Evaluate]
set textfield [lcdui.textfield -text {[lcdui.alert -text "hi!"] setcurrent}]
set form [lcdui.form -title "Text Field Example" -commandaction \
    [list runCode $textfield]]
$form append $textfield
$form setcurrent
$form addcommand $evalcmd

proc runCode {txtfld cmd frm} {
    eval [$txtfld cget -text]
}
```

Live example: http://www.heclbuilder.com/scripts/show/154

Name

lcdui.ticker — Creates a ticker that scrolls horizontally.

Synopsis

`lcdui.ticker` {-text *text*}

Description

The **lcdui.ticker** command creates a ticker that runs horizontally across the screen. For an in-depth look at the Java code that this command is based on, see: javax.microedition.lcdui.TextBox [http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/TextBox.html]

Example

```
set ticker [lcdui.ticker -text "The time is [clock format [clock time]]"]
set form [lcdui.form -title "Ticker Example" -ticker $ticker]
$form setcurrent

proc updateTicker {ticker} {
    $ticker configure -text "The time is [clock format [clock time]]"
    after 1000 [list updateTicker $ticker]
}

after 1000 [list updateTicker $ticker]
```

Live example: http://www.heclbuilder.com/scripts/show/155

Name

lcdui item options — common options for lcdui.choicegroup, lcdui.date, lcdui.gauge, lcdui.imageitem, lcdui.spacer, and lcdui.stringitem

Common lcdui item options

The lcdui "item" elements have a number of options in common, which are described here.

Many of these options are layout options, and correspond to what is described here: http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Form.html#layout

lcdui item configuration options

These options may be set as follows, or as options at the time the item is created.

`$itemcmd` {configure} {`-`*`optname`*} {*`optvalue`*}

- `-label` *`text`*: the item's label
- `-anchor` *`position`*: One of the following positions. For more information, see http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Graphics.html
  - `n`: top/center
  - `ne`: top/right
  - `e`: right/vertically centered
  - `se`: bottom/right
  - `s`: bottom/center
  - `sw`: bottom/left
  - `w`: left/vertically centered
  - `nw`: top/left
  - `center`: horizontally and vertically centered
  - `default`: top/left
  - `bl`: left/baseline
  - `bc`: centered/baseline
  - `br`: right/baseline
- `-shrink` *`1`* `|` *`0`* Indicate that this item's width may be reduced to its minimum width.
- `-expand` *`1`* `|` *`0`* Indicate that this item's width may be increased to fill available space. width.
- `-vexpand` *`1`* `|` *`0`* Indicate that this item's height may be increased to fill available space. width.
- `-newlinebefore` *`1`* `|` *`0`* Indicate that the next item (if any) in the container should be placed on a new line or row.

- `-newlineafter` *`1 | 0`* Indicate that this item should be placed at the beginning of a new line or row.
- `-layout2` *`1 | 0`* A layout directive indicating that new MIDP 2.0 layout rules are in effect for this item.
- `-preferredwidth` *`width`* Preferred width for the item.
- `-preferredheight` *`height`* Preferred height for the item.
- `-defaultcommand` *`command`* If this item is clicked on, the default command to call. *`command`* refers to a command object, and not a Hecl proc or command.
- `-commandaction` *`command`* The Hecl command to call when a command is dispatched.

read-only configuration information

All of the above attributes may be read as well as written. There are several more attributes that are "read only". They can be obtained like so:

`$itemcmd` {cget} {`-`*`optname`*} {*`optvalue`*}

- `-minwidth` : Returns the minimum width for this item.
- `-minheight` : Returns the minimum height for this item.

Name

midlet.exit — Exit from the Midetl

Synopsis

```
midlet.exit
```

Description

Exit from the current Midlet by calling `notifyDestroyed`

Name

midlet.pause — Pause the current midlet

Synopsis

```
midlet.pause
```

Description

Pause the current midlet by calling the `notifyPaused` method.

Name

midlet.resume — Resume execution of the current midlet

Synopsis

```
midlet.resume
```

Description

Attempts to resume execution of the midlet by calling the `resumeRequest` method.

Name

midlet.onpause — Hook to call when midlet execution is paused

Synopsis

```
midlet.onpause
```

Description

If this command is defined in the interpreter, it is called before the midlet is paused.

Name

midlet.onresume — Hook to call when midlet execution is resumed

Synopsis

```
midlet.onresume
```

Description

If this command is defined in the interpreter, it is called when execution of the midlet is resumed.

Name

midlet.checkpermissions — Query permission information

Synopsis

`midlet.checkpermissions` {*`permission_name`*}

Description

This command returns information about whether a given permission has been granted to the midlet. Quoting from the Java documentation:

> Get the status of the specified permission. If no API on the device defines the specific permission requested then it must be reported as denied. If the status of the permission is not known because it might require a user interaction then it should be reported as unknown.

The command returns 0 if the permission is denied; 1 if the permission is allowed, and -1 if the status is unknown

Example

```
midlet.checkpermissions "javax.microedition.io.Connector.file.read"
```

Name

midlet.getappproperty — Get application properties

Synopsis

`midlet.getappproperty` {*propertyname*}

Description

Lets the midlet retrieve application properties, as defined in the .jad file.

See also: system.getproperty

Name

midlet.platformrequest — Perform platform-specific actions

Synopsis

`midlet.platformrequest` {*url*}

Description

From the Javadocs:

> Requests that the device handle (for example, display or install) the indicated URL.
>
> If the platform has the appropriate capabilities and resources available, it SHOULD bring the appropriate application to the foreground and let the user interact with the content, while keeping the MIDlet suite running in the background. If the platform does not have appropriate capabilities or resources available, it MAY wait to handle the URL request until after the MIDlet suite exits. In this case, when the requesting MIDlet suite exits, the platform MUST then bring the appropriate application (if one exists) to the foreground to let the user interact with the content.

In other words, depending on the device, you can launch applications with midlet.platformrequest. For example:

```
# Make a phone call
midlet.platformrequest "tel:+393488866859"
# Start the sms sending application
midlet.platformrequest "sms:+393488866859"
# Open a web page
midlet.platformrequest "http://www.hecl.org"
# Install the jad/jar
midlet.platformrequest "http://www.hecl.org/jars/cldc1.1-midp2.0/Hecl.jad"
```

Note

There is not much standard about this command. It may or may not support different url types on different phones. Making phone calls with "tel:" URL's should always work. Also, keep in mind that on some devices, launching an external application may terminate the currently running midlet.

Name

midlet.resourceasstring — Get a resource file as a string

Synopsis

`midlet.resourceasstring` {*resourcename*}

Description

Given a resource *`resourcename`*, returns it as a string. This could be used, for instance, to load additional Hecl files.

Example

```
eval [midlet.resourceasstring "./more_commands.hcl"]
```

Name

midlet.flashbacklight — Flash the device's backlight

Synopsis

`midlet.vibrate` {*milliseconds*}

Description

Flashes the device's backlight for N milliseconds. Returns 1 if the light can be controlled by the application, and the display is in the foreground; otherwise, 0.

Name

midlet.vibrate — Vibrate the device

Synopsis

`midlet.vibrate` {*milliseconds*}

Description

Activates the device's vibrator for N milliseconds. Returns 1 if the vibrator can be controlled by the application and the display is in the foreground; otherwise 0.

Hacking Hecl's Java ME code

Since Java ME comes in several flavors that Hecl can be compiled for, it's necessary to understand what Hecl does, and how it does it.

JavaME has two layers that are of interest to us, "CLDC" and "MIDP". CLDC is available in 1.0 and 1.1 configurations, whereas MIDP comes in 1.0 and 2.0 configurations. The most common configurations are CLDC 1.0 and MIDP 1.0, CLDC 1.0 and MIDP 2.0, and CLDC 1.1 with MIDP 2.0. Here are the Wikipedia entries describing CLDC [http://en.wikipedia.org/wiki/Connected_Limited_Device_Configuration] and MIDP [http://en.wikipedia.org/wiki/MIDP].

Hecl tries to match code to system resources: in other words, the code in the `midp10/` and `midp10gui` (MIDP 1.0) directories is smaller, simpler, and has fewer features than the code in `midp20` and `midp20gui` (MIDP 2.0), reflecting the fact that many 1.0 devices only allow very small jar files ("midlets").

For MIDP 1.0, the `midp10gui` directory contains the `GUICmds.java`, which has most of the functionality that maps J2ME functionality to Hecl and back. The `midp10/Hecl.java` file contains the code that starts up Hecl on the cell phone. For MIDP 2.0, the `midp20gui` directory contains the GUI commands, and `midp20/Hecl.java` is where the application is launched from on the phone.

In order to be able to deal with all these different versions, Hecl is more or less forced to utilize a Java preprocessor, which explains all the ifdef's in the code. The various symbols are defined in the `settings.xml` file.

To compile different combinations of things, Hecl makes a couple of property files available that are used like so:

```
ant -propertyfile ./cldc10midp10.properties midlet
```

Which compiles the CLDC 1.0 / MIDP 1.0 version of Hecl and places the jar in the `jars/cldc1.0-midp1.0/` directory, or:

```
ant -propertyfile ./cldc11midp20.properties midlet
```

Which compiles the CLDC 1.1 / MIDP 2.0 version, and places the jar in the `jars/cldc1.1-midp2.0/` directory.

Hecl and BlackBerry

The BlackBerry platform allows you to utilize Java ME code, which is what we do to make Hecl run on that platform. This means that almost all of the Java ME commands are available, plus additional commands specific to the BlackBerry environment.

BlackBerry Commands

Name

browser.open — Launches the BlackBerry browser

Synopsis

`browser.open` {*url*} []

Description

The **browser.open** command launches the BlackBerry browser. It does not block while the browser remains open, so the execution of the Hecl script continues.

Name

device.systemversion — Returns a string with the system version information

Synopsis

```
device.systemversion
```

Description

Returns a string containing the system version of the device.

Note

This always returns an empty string on the emulator

Name

invoke.call — Makes a phone call using the built-in dialer

Synopsis

`invoke.call` {*phone_number*}

Description

Makes a call to the specified phone number, using the built in "dial" application.

Name

invoke.calculator — Opens the built in calculator application

Synopsis

```
invoke.calculator
```

Description

Launches the calculator application.

Name

invoke.camera — Starts the camera application

Synopsis

```
invoke.camera
```

Description

Launches the camera application.

Name

invoke.video — Starts the video application

Synopsis

```
invoke.video
```

Description

Launches the video application.

Name

servicebook.records — Retrieves connection information about various connection types

Synopsis

```
servicebook.records
```

Description

This call returns a list of all "service records". Each record is a hash with the following fields defined. For their meanings, please refer to this link: http://www.blackberry.com/developers/docs/4.7.0api/net/rim/device/api/servicebook/ServiceRecord.html

`APN` `ApplicationData` `BBRHosts` `BBRHosts` `BBRPorts` `BBRPorts` `CAAddress` `CAPort` `CARealm` `CID` `UID` `CidHash` `CompressionMode` `DataSourceId` `Description` `DisabledState` `EncryptionMode` `HomeAddress` `Id` `KeyHashForService` `LastUpdated` `Name` `NameHash` `NetworkAddress` `NetworkType` `Source` `Type` `Uid` `UidHash` `UserId` `isDirty` `isDisabled` `isEncrypted` `isInvisible` `isRecordProtected` `isRestoredFromBackup` `isRestoreDisabled` `isRestoreEnabled` `isSecureService` `isValid` `isWeakSecureService`

Hecl and Android

As of mid-2008, Hecl runs on Google's Android [http://code.google.com/android/] platform, although it's not yet a 'mature' port. That's ok right now, though, because Android isn't production ready yet, either.

Due to a different GUI model, a very extensive API, and a much more complete implementation of Java[1] Hecl on Android takes a different approach: the Android version of Hecl includes a **java** command, and introspection capabilities in order to be able to dynamically create Hecl commands that call out to native Java calls.

Android Hecl Quick Start

Developing for Android Hecl is quite similar to Java ME development, with the same cycle of editing a script, creating an application bundle, and testing it in an emulator.

1. To work with Android, of course the first thing you need to do is to get the SDK from Google: http://code.google.com/android/download.html. On my system, I installed it in here: `/opt/android-sdk/`.

2. Next, edit `android/android.properties` to point to the SDK, and the tools it needs to work (usually the `tools` directory within the SDK directory).

3. As with Java ME Hecl, you need a script file to work with. Here's a "hello world":

```
set context [activity]
set layout [linearlayout -new $context]
$layout setorientation VERTICAL
set layoutparams [linearlayoutparams -new {FILL_PARENT WRAP_CONTENT}]

set tv [textview -new $context -text {Hello World} \
```

[1] Android doesn't actually run Java, it just compiles code written in Java into bytecodes for the Dalvik engine.

```
                    -layoutparams $layoutparams]

$layout addview $tv
[activity] setcontentview $layout
```

To see more code, have a look at `android/res/raw/script.hcl`, which has examples of many different widgets.

4. You won't need it right away, but you might as well start the emulator:

   ```
   /opt/android-sdk/tools/emulator -avd your_avd
   ```

   ### Note

   You need to create an "AVD" before launching the emulator. This is documented here: http://developer.android.com/guide/developing/tools/emulator.html#avds

5. Now we create the new `Hello.apk` file.

   ```
   java -jar ./hecl/jars/AndroidBuilder.jar -android \
       /opt/android-sdk/platforms/android-1.5/ -class Hello \
       -label Hello -package hello.world \
       -script hello.hcl
   ```

   ### Note

   It's important to point out that the directory used with the `-android` option is a subdirectory of the SDK: `/opt/android-sdk/platforms/android-1.5/`, rather than the top level.

   The command line options are as follows:

   `-android`: The location of the Android SDK.
   `-class`: The class name to utilize for the new .apk.
   `-label`: The user-visible name of the package.
   `-package`: The Java package to use. You can pretty much use any name you like, it doesn't matter much.
   `-script`: The location of the script you want to use in the new .apk.

6. We now have a `Hecl.apk` file. We need to sign it with the debug key:

   ```
   keytool -genkeypair -keystore debug.keystore \
       -keypass android -alias androiddebugkey -storepass android \
       -dname "CN=Android Debug,O=Android,C=US"

   jarsigner -keystore debug.keystore -keypass android \
       -storepass android -verbose Hello.apk androiddebugkey
   ```

7. At this piont, we can send the signed file, `Hello.apk` to the emulator:

   ```
   /opt/android-sdk_m5-rc15_linux-x86/tools/adb install Hello.apk
   ```