; Documentation for Questor

Functions

#begin_function
abs(x)
Returns the absolute value of the number, i.e. without the minus sign.
Ex:
abs(23)
returns **23**
abs(-23)
returns **23**
#end_function

#begin_function
alert_user([title]; [message]; [button1]; [button2]; [button3])
Brings up an alert panel with one, two or three buttons. All the arguments are optional.
title is the title at the top of the panel.
message is the message in the panel.
button1 is the title of the button to the right.
button2 is the title of the button in the middle.
button3 is the title of the button to the left.
- If the button to the **right** is pressed, the function returns *true*.
- If the button in the **middle** (with three buttons) is pressed, the function returns *false*.
- If the button to the **left** (with three buttons) is pressed, the function returns *nil*.

Ex:
```
alert_user("Note!"; "You can not do this!")

x := alert_user("Alert"; "Wanna hear a beep?"; "Beep"; "No")
if x = true then
        beep()
end if
```
#end_function

#begin_function
acos(x)
Returns the acos of a factor.
Ex:
acos(0.6)
returns **0.92729521800161**
#end_function

#begin_function
asin(x)
Returns the asin of a factor.
Ex:
asin(0.6)
returns **0.64350110879328**
#end_function

#begin_function
atan(x)
Returns the atan of a factor.
Ex:
atan(1.0)
returns **0.78539816339745**
#end_function

#begin_function
atan2(x; y)
Returns the atan of **x / y**.
Ex:
atan2(2; 3)
returns **0.58800260354757**
#end_function

#begin_function
avg(args...)
Returns the average of the list of arguments.
Ex:
avg(12; 13; 17)
returns **14.**
avg(A1:G12)
returns the average of the values in the cell range A1:G12
#end_function

#begin_function
bailout()
Programmatic interruption of QScript execution. Equivalent to a user interrupt by pressing **Command-.**
Does not return a value.
#end_function

#begin_function
beep()
Plays the system beep.
#end_function

#begin_function
break_off()
Disables the possibility for the user to break QScript execution by pressing **Command-.**
Also see **break_on()** and **set_break_enabled()**
Note: If the QScript code goes into an infinite loop, Questor will hang forever.
Do **not** do this:

```
break_off()
while true do
        ; something
end while
```

#end_function

#begin_function
break_on()
Enables the possibility for the user to break QScript execution by pressing **Command-.**
Also see **break_off()** and **set_break_enabled()**
#end_function

#begin_function
call_doc_handler(handler; args...)
Calls a document handler.
handler is the name of the handler.
args are the arguments to the handler.
Returns the value that is returned by the handler.
Ex:
call_doc_handler("document_was_hidden")
will execute the **document_was_hidden** handler in the document.

Assume you have created a handler in the document that is called **my_handler(x; y)** that returns x + y.
call_doc_handler("my_handler"; 12; 34)
will then return **46**
#end_function

#begin_function
call_function(function; args...)
Calls a function.
function is the name of the function.
args are the arguments to the function.
Returns the value that is returned by the function.
Ex:
call_function("cos"; 0.7)
returns **0.76484218728449**

#end_function

#begin_function
call_gadget_handler(gadget; handler; args...)
Calls a gadget handler.
gadget is the name or number of the gadget.
handler is the name of the handler.
args are the arguments to the handler.
Returns the value that is returned by the handler.
Ex:
call_gadget_handler(1; "action")
will execute the **action** handler in button 1.

Assume you have named the button to **my_button.**
call_gadget_handler("my_button"; "action")
will execute the **action** handler in button **my_button.**
#end_function

#begin_function
call_remote_doc_handler(document; handler; args...)
Calls a document handler in another worksheet.
document is the name of the document.
handler is the name of the handler.
args are the arguments to the handler.
Returns the value that is returned by the handler.
Ex:
call_remote_doc_handler("doc2";"document_was_hidden")
will execute the **document_was_hidden** handler in the document **doc2.**
#end_function

#begin_function
call_remote_function(document; function; args...)
Calls a function in another worksheet.
document is the name of the document.

function is the name of the function.
args are the arguments to the function.
Returns the value that is returned by the function.
Ex:
call_remote_function("doc2"; "cos"; 0.7)
will call the function **cos** in the document **doc2** and return **0.76484218728449**
#end_function

#begin_function
capitalize(string)
Capitalizes the first letter of **string**. No other characters are affected.
Ex:
capitalize("hello world")
returns **Hello world**
#end_function

#begin_function
case_exact(string1; string2)
Compares two strings and returns *true* if the strings are equal. This function is **not** case sensitive.
Also see **exact()**
Ex:
case_exact("abc"; "abc")
returns *true*
case_exact("abc"; "abC")
returns *true*
case_exact("abc"; "abd")
returns *false*
#end_function

#begin_function
cbrt(x)
Returns the cubic root of x.
Ex:

cbrt(125)
returns **5**
#end_function

#begin_function
ceiling(x)
Returns the smallest integer no less than the number.
Ex:
ceiling(2.5)
returns **3**
ceiling(-2.5)
returns **-2**
#end_function

#begin_function
char(number)
Returns the character with the ASCII code **number.**
Ex:
char(65)
returns **A**
#end_function

#begin_function
clean(string)
Removes control characters from the string. These characters are usually printing and format control codes used by word processors to control the format of the text.
#end_function

#begin_function
clear_console()
Removes all old text in the console and the command history list.
#end_function

#begin_function
close([path])
Closes a file that has been opened with the **open()** function.
If only one file is opened, **path** can be omitted.
If more that one file is opened, **path** can be either the full path name, the full file name or just the file name without extension.
Also see **open()**
Ex:
If the file **/tmp/my_file.txt** has been opened, all the following expressions will close it:
close("/tmp/my_file.txt")
close("my_file.txt")
close("my_file")
close() ; *if it is the only open file*
#end_function

#begin_function
code(char)
Returns the ASCII value of the first character of the string **char**.
Ex:
code("A")
returns **65**
#end_function

#begin_function
confirm_user([title]; [message])
Brings up an alert panel with two buttons. All the arguments are optional.
title is the title at the top of the panel.
message is the message in the panel.
- If the **OK** button is pressed, the function returns *true*.
- If the **Cancel** button is pressed, the function returns *false*.
Ex:
x := confirm_user("Alert"; "Wanna hear a beep?")

```
if x = true then
        beep()
end if
#end_function
```

```
#begin_function
```
cos(x)

Returns the cosinus of the angle x. The angle must be expressed in radians.
Ex:
cos(0.7)
returns **0.76484218728449**
```
#end_function
```

```
#begin_function
```
cterm(interest; future_value; present_value)

Calculates the number of periods required for an investment to grow from a present value to a future value by compounding the interest.
The function uses the following formula:

cterm = ln(future_value / present_value) / ln(1 + interest)

where **ln()** is the natural logarithm.
Ex:
cterm(0.18; 2000; 1000)
returns **4.18783513351232**
cterm(0.015; 2000; 1000)
returns **46.55552563080618**
```
#end_function
```

```
#begin_function
```
date(year; [month]; [day])

Returns a date value from **year**, **month** and **day**.
Ex:
date(1991)

returns **01-Jan-91**
date(1991; 3)
returns **01-Mar-91**
date(1991; 3; 12)
returns **12-Mar-91**
#end_function

#begin_function
datevalue(string)
Converts a string into a date value.
Ex:
datevalue("12-Aug-1992")
returns **12-Aug-1992**
datevalue("12/12/86")
returns **12/12/86**
#end_function

#begin_function
date_to_number(date)
Converts a date value (or a time value) into a number.
The integer part of the number is the number of days since **31-Dec-1899.**
The decimal part of the number is the time of the day.
Also see **number_to_date()**
Ex:
date_to_number(1-Jan-1900)
returns **1**
date_to_number(1991-08-12)
returns **33461**
date_to_number(6:00:00)
returns **0.25**
date_to_number(11:00:03)
returns **0.45836805555556**
#end_function

#begin_function
date_add_days(date; [no_of_days])
Adds **no_of_days** to **date.**
Ex:
date_add_days(93-12-12; 10)
returns **93-12-22**
date_add_days(12-Dec-93; 20)
returns **01-Jan-94**
#end_function

#begin_function
date_add_months(date; [no_of_months])
Adds **no_of_months** to **date.**
Ex:
date_add_months(93-11-12; 1)
returns **93-12-12**
date_add_months(12-Dec-93; 2)
returns **12-Feb-94**
#end_function

#begin_function
date_add_years(date; [no_of_years])
Adds **no_of_years** to **date.**
Ex:
date_add_years(91-11-12; 1)
returns **92-11-12**
date_add_years(12-Sep-89; 2)
returns **12-Sep-91**
#end_function

#begin_function
date_subtract_days(date; [no_of_days])
Subtracts **no_of_days** from **date.**
Ex:

date_subtract_days(92-01-23; 4)
returns **92-01-19**
date_subtract_days(12-Jan-87; 15)
returns **28-Dec-86**
#end_function

#begin_function
date_subtract_months(date; [no_of_months])
Subtracts **no_of_days** from **date.**
Ex:
date_subtract_months(92-03-23; 2)
returns **92-01-23**
date_subtract_months(12-Jan-87; 3)
returns **12-Oct-86**
#end_function

#begin_function
date_subtract_years(date; [no_of_years])
Subtracts **no_of_days** from **date.**
Ex:
date_subtract_years(92-03-23; 2)
returns **90-03-23**
date_subtract_years(12-Jan-87; 3)
returns **12-Jan-84**
#end_function

#begin_function
day(date)
Returns the day of the month for a date.
Also see **year()**, **month()**, **week_day()** and **year_day()**
Ex:
day(12-Aug-1987)
returns **12**
day(1992-12-11)

returns **11**
#end_function

#begin_function
directory_for_file(path)
Returns the directory for the file **path.** No control is made if path is a valid filename.
Ex:
directory_for_file("/LocalLibrary/My_File.xqr")
returns **/LocalLibrary**
#end_function

#begin_function
dragged_files()
Returns the file names of the files that are currently dragged. The file names will be separated by tabs.
This function is usually used in the handlers of a file-well to check the dragged files.
Also see **dragged_file_no()**
#end_function

#begin_function
dragged_file_no(index)
Returns the file name of file **index** in the files that are currently dragged.
This function is usually used in the handlers of a file-well to check the dragged files.
Also see **dragged_files()**
#end_function

#begin_function
d360(date1; date2)
Returns the number of days between **date1** and **date2.**
Ex:

d360(93-01-01; 93-02-01)
returns **-31**
d360(04/17/87; 04/12/87)
returns **5**
d360(01/24/88; 12/24/87)
returns **31**
#end_function

#begin_function
err()
Returns the generic error value **#error**
#end_function

#begin_function
error_number(error_code)
Returns the error value for **error_code.**

error_code	**error value**
1	#divide_by_zero
2	#undefined_name
3	#undefined_function
4	#too_few_arguments
5	#too_many_arguments
6	#illegal_argument
7	#unbound_local
8	#unbound_global
9	#bad_next_loop
10	#bad_exit_loop
11	#broken_function
12	#bad_return
13	#stack_overflow
14	#invalid_expression
15	#unsupported_feature
16	#error
17	#number_out_of_bounds

18 #invalid_file
19 #undefined_handler
20 #invalid_gadget_name
21 #circular_reference
22 #bad_external_reference
23 #range_out_of_bounds
24 #invalid_define_macro
25 #invalid_criteria
#end_function

#begin_function
eval_in_unix_shell(command_string)
Evaluates **command_string** in a UNIX shell and returns the output (stdout) as a string.
Warning: Do not use the **eval_in_unix_shell** function to start programs that takes over the terminal and waits for input (like **bc**). This will lead to an infinite loop.
Also see **system()**
Ex:
eval_in_unix_shell("date")
returns **Wed Feb 24 12:21:58 GMT+0100 1993**
#end_function

#begin_function
exact(string1; string2)
Compares two strings and returns *true* if the strings are equal. This function is case sensitive.
Also see **case_exact()**
Ex:
case_exact("abc"; "abc")
returns *true*
case_exact("abc"; "abC")
returns *false*
case_exact("abc"; "abd")
returns *false*

#end_function

#begin_function
exp(x)
Returns *e* raised to x.
Ex:
exp(1)
returns **2.718282**
#end_function

#begin_function
false()
Returns the value *false*.
Also see **true()**
#end_function

#begin_function
filesize([path])
Returns the number of bytes in an open ASCII file.
path is the full name of the file.
If only one file is opened, **path** can be omitted.
If more that one file is opened, **path** can be either the full path name, the full file name or just the file name without extension.
Ex:
open("/tmp/my_file")
write("a string")
filesize()
returns **7**
close()
#end_function

#begin_function
file_exist(path)

Returns *true* if the file path exists. Otherwise the function returns *false*.
Ex:
file_exist("/NextApps/Edit.app")
returns *true*
#end_function

#begin_function
file_extension(path)
Returns the file extension for the file **path.** No control is made if path is a valid filename.
Ex:
file_extension("/NextApps/Edit.app")
returns **app**
#end_function

#begin_function
file_name(path)
Returns the file name for the file **path.** No control is made if path is a valid filename.
Ex:
file_name("/NextApps/Edit.app")
returns **Edit.app**
#end_function

#begin_function
find(substring; string; [offset])
Searches for the first occurrence of **substring** in **string.**The search starts offset characters from the beginning of the string. Returns the character position where the substring starts or false if there is no match.
Ex:
find("an"; "Can you dance?") ; *starts to search from "C"*
returns **2**
find("an"; "Can you dance?"; 2) ; *starts to search from "n"*
returns **10**

find("an"; "Can you dance?"; 11) ; *starts to search from "c"*
returns **false**
#end_function

#begin_function
floor(x)
Returns the largest integer not greater than x.
Ex:
floor(2.5)
returns **2**
floor(-2.5)
returns **-3**
#end_function

#begin_function
fv(payments; interest; term)
Computes the future value of a series of equal payments, at a specified interest rate, for a specified number of periods.
The function uses the following formula:

fv = payment * ((1 + interest)^term - 1) / interest

Ex:
fv(3600; 0.12; 5)
returns **22870.25**
fv(300; 0.01; 60)
returns **24500.90**
#end_function

#begin_function
gauge_value(name)
Returns the value of a gauge.
name is the name or number of the gauge.
Ex:

gauge_value(1)
returns the value of gauge 1.

Assume you have named the gauge to **my_gauge.**
gauge_value("my_gauge")
returns the value of gauge **my_gauge**.
#end_function

#begin_function
getpos([path])
Returns the position of the file pointer in a file opened by the **open()** function. The first character position in a file is zero.
If only one file is opened, **path** can be omitted.
If more that one file is opened, **path** can be either the full path name, the full file name or just the file name without extension.
Also see **setpos()**
Ex:
open("/tmp/my_file")
getpos()
returns **0**
write("a string")
getpos()
returns **8**
close()
#end_function

#begin_function
hour(time)
Returns the hour of the day of a time value.
Also see **minute()** and **second()**
Ex:
hour(7:30:00)
returns **7**
#end_function

#begin_function
if(condition; then_expr; else_expr)
Returns **then_expr** if **condition** evaluates to *true*. Otherwise the function returns **else_expr**.
Ex:
if(12 < 13; 100; 200)
returns **100**
if(12 > 13; 100; 200)
returns **200**
#end_function

#begin_function
info(attribute)
Returns information about the version of Questor etc. **attribute** can be one of the following:
"directory" - returns the directory of the active document.
"filename" - returns the full file name of the active document.
"filename_only" - returns the file name of the active document.
"numfiles" - returns the number of open documents.
"release" - returns the version of the Questor application.
"patchlevel" - returns the patch level of the Questor application.
"system" - returns the hardware (currently **NeXT** or **Intel**).
#end_function

#begin_function
int(x)
Returns the integer part of x.
??? now same as floor!!!!
Ex:
int(2.5)
returns **2**
int(-2.5)
returns **-2**
#end_function

#begin_function
iserror(value)
Same as **is_error()**
Used for 1-2-3 function compatibility only.
#end_function

#begin_function
isnumber(value)
Same as **is_number()**
Used for 1-2-3 function compatibility only.
#end_function

#begin_function
isstring(value)
Same as **is_string()**
Used for 1-2-3 function compatibility only.
#end_function

#begin_function
is_boolean(value)
Returns *true* if **value** is *true* or *false*. Otherwise, it returns *false*.
#end_function

#begin_function
is_bound(value)
Returns *true* if the variable with the name **value** is bound to a value. Otherwise the function returns *false*.
Ex:
is_bound("x")
returns **false**
x := 12   ; *the variable x is bound to the value 12*

is_bound("x")
returns **true**
#end_function

#begin_function
is_considered_false(value)
Returns *true* if **value** is **0**. Otherwise, it returns *false*.
#end_function

#begin_function
is_considered_number(value)
Returns *true* if **value** is a number or a time or date value. Otherwise, it returns *false*.
#end_function

#begin_function
is_considered_true(value)
Returns *true* if **value** is **1**. Otherwise, it returns *false*.
#end_function

#begin_function
is_date(value)
Returns *true* if **value** is a time or date value. Otherwise, it returns *false*.
#end_function

#begin_function
is_defined(value)
Returns *true* if the function with the name **value** is defined. Otherwise the function returns *false*.
Ex:
is_defined("sum")
returns **true**

is_defined("my_sum")
returns **false**   *; unless you defined it yourself...*
#end_function

#begin_function
is_directory(path)
Returns *true* if **path** is a directory. Returns *false* if **path** is a file and *nil* if **path** does not exist.
Ex:
is_directory("/NextApps")
returns **true**
is_directory("/NextApps/Mail.app/Mail")
returns **false**
is_directory("/NextApps/Mail.app/Mial")
returns **nil**
#end_function

#begin_function
is_error(value)
Returns *true* if **value** is an error value. Otherwise, it returns *false*.
#end_function

#begin_function
is_false(value)
Returns *true* if **value** is *false*. Otherwise, it returns *false*.
#end_function

#begin_function
is_globally_bound(value)
Returns *true* if the variable with the name **value** is globally bound to a value. Otherwise the function returns *false*.
Note: All variables that are assigned to a value in the Console window will be global.

Ex:
is_globally_bound("x")
returns **false**
global x := 12   *; the global variable x is bound to the value 12*
is_globally_bound("x")
returns **true**
#end_function

#begin_function
is_locally_bound(value)
Returns *true* if the variable with the name **value** is locally bound to a value. Otherwise the function returns *false*.
Note: All variables that are assigned to a value in the Console window will be global. The example below only works within a function or handler.
Ex:
is_locally_bound("x")
returns **false**
x := 12   *; the local variable x is bound to the value 12*
is_locally_bound("x")
returns **true**
#end_function

#begin_function
is_nil(value)
Returns *true* if **value** is the *nil* value. Otherwise, it returns *false*.
#end_function

#begin_function
is_number(value)
Returns *true* if **value** is a number. Otherwise, it returns *false*.
#end_function

#begin_function

is_string(value)
Returns *true* if **value** is a string. Otherwise, it returns *false*.
#end_function

#begin_function
is_true(value)
Returns *true* if **value** is *true*. Otherwise, it returns *false*.
#end_function

#begin_function
launch_application(application; [host])
Launches **application** on the machine **host**. If host is omitted, the application will launch on the same machine.
Returns *true* if the function was successful. Otherwise the function returns *false*.
Ex:
All the following expressions will launch **Grab** on the same machine:
launch_application("/NextApps/Grab.app")
launch_application("Grab.app")
launch_application("Grab")
This will launch **Mail** on the machine **next2**:
launch_application("Mail", "next2")
#end_function

#begin_function
left(string; n)
Returns the **n** characters to the left in **string**.
Ex:
left("a string"; 4)
returns **a st**
#end_function

#begin_function

length(string)
Returns the number of characters in **string**.
Ex:
length("1234567890")
returns **10**
length("a string")
returns **8**
#end_function

#begin_function
ln(x)
Returns the base-e (natural) logarithm of **x**.
Ex:
ln(10)
returns **2.30258509299405**
#end_function

#begin_function
log(x)
Returns the base-10 logarithm of **x**.
Ex:
log(10)
returns **1**
#end_function

#begin_function
lower(string)
Converts all characters in **string** to lower case.
Also see **upper()**
Ex:
lower("hello, HELLO")
returns **hello, hello**
#end_function

#begin_function
match_extension(path; extension)
Returns true if **path** has the specified file extension. No control is made if path is a valid filename.
Ex:
match_extension("/NextApps/Edit.app"; "app")
returns **true**
match_extension("/NextApps/Edit.app"; "txt")
returns **false**
#end_function

#begin_function
max(args...)
Returns the maximum of the list of arguments.
Ex:
max(12; 13; 17)
returns **17.**
max(A1:G12)
returns the maximum value in the cell range A1:G12
#end_function

#begin_function
mid(string; [start]; [n])
Returns a substring of **string**, starting with character **start**, and **n** characters long. The first character is number 1.
Ex:
mid("Xanthus Questor"; 1; 7)
returns **Xanthus**
mid("Xanthus Questor"; 9; 8)
returns **Questor**
mid("Xanthus Questor")
returns **X**
#end_function

#begin_function
min(args...)
Returns the minimum of the list of arguments.
Ex:
min(12; 13; 17)
returns **17.**
min(A1:G12)
returns the minimum value in the cell range A1:G12
#end_function

#begin_function
minute(time)
Returns the minute of the hour of a time value.
Also see **hour()** and **second()**
Ex:
minute(7:30:00)
returns **30**
#end_function

#begin_function
mod(x; y)
Returns **x** modulo **y**.
Ex:
mod(10; 12)
returns **10**
mod(14; 12)
returns **2**
#end_function

#begin_function
month(date)
Returns the month of the year for a date.
Also see **year() day()**, **week_day()** and **year_day()**

Ex:
month(12-Aug-1987)
returns **8**
month(1992-12-11)
returns **12**
#end_function

#begin_function
month_day_count(date)
Returns the number of days of the month in **date.**
Ex:
month_day_count(92-02-11)
returns **29**
month_day_count(12-Aug-83)
returns **31**
#end_function

#begin_function
nil()
Returns the value *nil*.
#end_function

#begin_function
now()
Returns a time value for the time of the computer clock.
Ex:
now()
returns something like **10:27:58**
date_to_number(now())
returns something like **0.43616898148148**
#end_function

#begin_function

npv(interest; values...)
Computes the net present value of a series of future periodic cash flows, discounted at a fixed periodic interest rate. The function assumes that each flow in the range is made at the end of each period.
The function uses the following formula:

npv = sum(Vi / (1 + interest)^i)

where **Vi** = the series of cash flows found in **values**, and **i** = the number of the specific cash flow i the range.
Ex:
Assume the range **A1:A6** looks like this:
3 000.00
3 000.00
-2 500.00
3 000.00
3 500.00
1 000.00

npv(0.1; A1:A6)
then returns **8115.06**
#end_function

#begin_function
number_of_dragged_files()
Returns the number of files that are currently dragged. This function is usually used in the handlers of a file-well to check the dragged files.
Also see **dragged_files()**
#end_function

#begin_function
number_to_date(number)
Converts a number to a date value.
Also see **date_to_number()**
Ex:

number_to_date(32000)
returns **12-Aug-87**
date_to_number(12-Aug-87)
returns **32000**
#end_function

#begin_function
number_to_time(number)
Converts a number to a time value.
Also see **date_to_number()**
Ex:
number_to_time(0.5)
returns **12:00:00**
number_to_time(0.75)
returns **18:00:00**
number_to_time(0.89)
returns **21:21:36**
#end_function

#begin_function
open(path; [mode])
Opens a new or an old file for reading or writing ASCII text.
path is the complete filename for the file.
mode specifies how the file will be accessed. If it is omitted, the file will be opened in **write** ("w") mode.
"r" - opens and reads a file that already exists.
"m" - opens a modifies a file that already exists. You can use all the file commands, including **write()** and **writeln()**.
"w" - creates a new file (erasing any file with that name). You can use all the file commands, including **write()** and **writeln()**.
"a" - opens an existing file with the file pointer at the end of the file. You can use all the file commands.
More than one file can be opened at the same time.
Also see **close()**
Ex:

open("/tmp/new_file", "w")
creates a new file.
open("/tmp/old_file", "r")
opens the file **/tmp/old_file** for reading.
#end_function

#begin_function
open_alien_file(file; [application])
Opens **file** with **application.** If **application** is omitted, the file will be opened with its default application.
Returns *true* if the function was successful. Otherwise the function returns *false*.
Ex:
open_alien_file("/etc/hosts")
will open the file in **Edit.**
open_alien_file("/etc/hosts"; "WriteNow")
will open the file in **WriteNow.**
#end_function

#begin_function
pi()
Returns the constant **pi.**
Ex:
pi()
returns **3.1415926536**
#end_function

#begin_function
play_sound_file(file)
Plays a sound file. Returns *true* if successful. Otherwise, it returns *false*.
Ex:
play_sound_file("/NextLibrary/Sounds/Glass.snd")
#end_function

#begin_function
pmt(principal; interest; term)
Calculates the amount that is to be paid each period to repay a loan.
The function uses the following formula:

pmt = principal * (interest / 1 - (interest + 1)^-term)

where **principal** is the face amount of the loan, **interest** is the periodic interest rate and **term** is the number of payments.
Ex:
pmt(5000; 0.18; 10)
returns **1112.57**
#end_function

#begin_function
print(value)
Prints **value** in the Console window.
#end_function

#begin_function
prompt_for_date([title])
Opens a panel where you can enter a date or a time value. **title** is the title of the panel.
Returns the date or time value that was entered in the panel. If nothing was entered or if a non-valid date or time value was entered, it returns the error value **#error**.
#end_function

#begin_function
prompt_for_number([title])
Opens a panel where you can enter a number. **title** is the title of the panel.
Returns the number that was entered in the panel. If nothing was entered or if a non-valid number was entered, it returns the error value **#error**.

#end_function

#begin_function
prompt_for_string([title])
Opens a panel where you can enter a string. **title** is the title of the panel.
Returns the string that was entered in the panel.
#end_function

#begin_function
proper(string)
Converts all characters in **string** to lower case. The first character of every word is converted to upper case.
Ex:
proper("hello world")
returns **Hello World**
proper("HELLO WORLD")
returns **Hello World**
#end_function

#begin_function
protect(expr)
Protects an expression **expr** so that a run-time error does not occur. If an error occurs, the function will just return the error value.
Ex:
protect(12 / 0)
returns **#divide_by_zero**
#end_function

#begin_function
pv(payment; interest; term)
Calculates the present value of an annuity (a series of equal payments) discounted at an interest rate over a number of periods. The function assumes each payment is made at the end of each time period.

The function uses the following formula:

pv = payment * (1 - (1 + interest)^-term) / interest

where **payment** is the periodic payment, **interest** is the periodic interest rate and **term** is the number of periods.
Ex:
pv(100000; 0.1; 15)
returns **760607.95**
#end_function

#begin_function
rand()
Returns a random number between 0 and 1.
Ex:
rand()
returns something like **0.31542536616325**
#end_function

#begin_function
random(x; y)
Returns a random number between (and including) **x** and **y**.
Ex:
random(10; 20)
returns something like **13**
#end_function

#begin_function
rate(future_value; present_value; term)
Calculates the interest rate required for a current investment to grow to a future value over a set term, if the interest is compounded.
The function uses the following formula:

rate = (future_value / present_value)^(1 / term) - 1

Ex:
rate(2000; 1000; 5)
returns **0.1487**
#end_function

#begin_function
read([count]; [path])
Returns a portion of an ASCII file that is opened with the **open()** function. The function will start reading from the current position of the file pointer. The file pointer then moves to the first character after those that are read.
count is the number of characters that should be read.
path is the file name of the file.
If **count** is omitted, the function will read one character.
If only one file is opened, **path** can be omitted.
If more that one file is opened, **path** can be either the full path name, the full file name or just the file name without extension.
Also see **open()**, **close()** and **readln()**
Ex:
open("/tmp/oldfile", "r")
first_chars := read(25)
close()
#end_function

#begin_function
readln([path])
Returns a portion of an ASCII file that is opened with the **open()** function. The function will start reading from the current position of the file pointer to the next carriage return. The file pointer then moves to the first character after those that are read.
If only one file is opened, **path** can be omitted.
If more that one file is opened, **path** can be either the full path name, the full file name or just the file name without extension.
Also see **open()**, **close()** and **read()**
Ex:

open("/tmp/oldfile", "r")
first_line := readln()
close()
#end_function

#begin_function
ref_count(value)
Returns the reference count for **value.**
Ex:
ref_count(A1)
will return the number of references to the cell **A1.**
#end_function

#begin_function
repeat(string; [n])
Returns a string with **string** repeated n times.
Ex:
repeat("hello world")
returns **hello world**
repeat("hello world"; 2)
returns **hello worldhello world**
#end_function

#begin_function
replace(string; [start]; [n]; [new_string])
Replaces **n** characters in **string** with **new_string**, starting with character **start.**
The first character is number **1.**
Ex:
replace("Hello world!"; 1; 5; "Goodbye")
returns **Goodbye world!**
replace("Hello world!"; 7; 5; "Jim")
returns **Hello Jim!**
#end_function

#begin_function
right(string; n)
Returns the **n** characters to the right in **string.**
Ex:
right("a string"; 4)
returns **ring**
#end_function

#begin_function
round(x)
Returns the nearest integer to x.
Ex:
round(2.3)
returns **2**
round(2.7)
returns **3**
#end_function

#begin_function
second(time)
Returns the second of the minute of a time value.
Also see **hour()** and **minute()**
Ex:
second(7:30:23)
returns **23**
#end_function

#begin_function
select_string([title]; [list_title]; [selected]; strings...)
Opens a panel with a selection list.
title is the title of the panel.
list_title is the title of the list.
selected is the string that should be selected by default.

strings are the strings in the list.
If you click **OK** in the panel, it returns the selected string.
If you click **Cancel** in the panel, it returns *false*.
Ex:
select_string(; ; ; "One"; "Two"; "Three")
select_string("Pizza Place Menu"; "What size of Pizza do you want?"; "Regular"; "Small"; "Regular"; "Big")
#end_function

#begin_function
send_alien_message(host; application; selector; arg_types; args...)
Sends an Objective-C message to another NEXTSTEP application. You can send arguments or pointers to arguments to get return values.
host is the machine that runs the application. If **host** is *nil* it means the local machine.
application is the name of the NEXTSTEP application.
selector is the name of the Objective-C message you want to send.
arg_types is a string that specifies the types of the arguments to the message.
args are the arguments to the message.
The different argument types are:
i - integer
d - double (float)
b - boolean
c - string
If the argument is a pointer to a value, use capital letters.
I - pointer to an integer
D - pointer to a double (float)
B - pointer to a boolean
C - pointer to a string
Returns *true* if the function was successful. Otherwise the function returns *false*.
Ex:
The NEXTSTEP application **Edit** understands the Objective-C message
- openFile:(STR)file ok:(int *)flag;
To open the file */etc/hosts* in Edit you can type the following:
send_alien_message(nil, "Edit", "openFile:ok:", "cI", "/etc/hosts", "ok")

Note: Since the second argument is a pointer to an integer the argument in the **args** collection should be the name of a variable.
If the message was sent successfully, the variable **ok** will have the value **1.**
#end_function

#begin_function
set_break_enabled(boolean)
Disables or enables the possibility for the user to break QScript execution by pressing **Command-.**
Also see **break_on()** and **break_off()**
Note: If the QScript code goes into an infinite loop, Questor will hang forever.
Do **not** do this:

```
set_break_enabled(false)
while true do
        ; something
end while
```

#end_function

#begin_function
setpos(position; [path])
Places the file pointer at **position** in a file that has bee opened with the **open()** function. The first character position in a file is zero.
If only one file is opened, **path** can be omitted.
If more that one file is opened, **path** can be either the full path name, the full file name or just the file name without extension.
The function returns **false** if position is a non-existing character position.
Otherwise the function returns **true.**
Also see **open()**
Ex:
Assume the file **/tmp/my_file.txt** is opened.
Then all the following are correct:

```
setpos(2; "/tmp/my_file.txt")
setpos(2; "my_file.txt")
setpos(2; "my_file")
```

#end_function

#begin_function
set_gauge_value(name; value)
Sets the value of a gauge to **value**. **name** is the number or name of the gauge. Returns *true* if successful.
Ex:
set_gauge_value(2; 23)
returns **true**
set_gauge_value("my_gauge"; 43)
returns **true**
set_gauge_value("my_gauge"; 12)
returns **#invalid_gadget_name**   ; *if my_gauge does not exist*
#end_function

#begin_function
set_slider_value(name; value)
Sets the value of a slider to **value**. **name** is the number or name of the slider. Returns *true* if successful.
Ex:
set_slider_value(2; 23)
returns **true**
set_slider_value("my_slider"; 43)
returns **true**
set_slider_value("my_slider"; 12)
returns **#invalid_gadget_name**   ; *if my_slider does not exist*
#end_function

#begin_function
set_textfield_value(name; value)
Sets the value of a text-field to **value**. **name** is the number or name of the text-field. Returns *true* if successful.
Ex:
set_textfield_value(2; 23)
returns **true**

set_textfield_value("my_text"; 43)
returns **true**
set_textfield_value("my_text"; 12)
returns **#invalid_gadget_name** ; *if my_text does not exist*
#end_function

#begin_function
set_timer_enabled(name; value)
Enables or disables a timer. **name** is the number or name of the timer. Returns *true* if successful.
Ex:
set_timer_enabled(3; true)
returns **true**
set_timer_enabled("my_timer"; false)
returns **true**
set_timer_enabled("my_timer"; true)
returns **#invalid_gadget_name** ; *if my_timer does not exist*
#end_function

#begin_function
set_timer_period(name; value)
Sets the period of a timer in seconds. **name** is the number or name of the timer. Returns *true* if successful.
Note: You must restart the timer for the new period to have any effect.
Ex:
set_timer_period(3; 2)
returns **true**
set_timer_period("my_timer"; 3)
returns **true**
set_timer_period("my_timer"; 4)
returns **#invalid_gadget_name** ; *if my_timer does not exist*
#end_function

#begin_function

sin(x)
Returns the sinus of the angle x. The angle must be expressed in radians.
Ex:
sin(0.7)
returns **0.64421768723769**
#end_function

#begin_function
slider_value(name)
Returns the value of a slider. **name** is the number or name of the slider.
Ex:
slider_value(2)
returns something like **65.85366129875183**
slider_value("my_slider")
returns something like **65.85366129875183**
slider_value("my_slider")
returns **#invalid_gadget_name**   ; *if my_slider does not exist*
#end_function

#begin_function
slide_file_icon(file; from_x; from_y; to_x; to_y)
Slides the file icon of **file** over the screen from the point **(from_x, from_y)** to point **(to_x, to_y)**. Always returns *true*. If the file does not exist, a "cannot find file"-icon will be used.
Ex:
slide_file_icon("/NextApps/Edit.app"; 0; 0; 1000; 1000)
returns **true**
#end_function

#begin_function
slide_image(file; from_x; from_y; to_x; to_y)
Slides the image in **file** over the screen from the point **(from_x, from_y)** to point **(to_x, to_y)**. Returns *true* if successful and *false* if **file** does not exist.
Ex:

slide_image("/NextApps/Preferences.app/loud.tiff"; 0; 0; 1000; 1000)
returns **true**
#end_function

#begin_function
sln(cost; salvage; life)
Calculates annual depreciation using the "straight line" method.
The function uses the following formula:

sln = (cost - salvage) / life

Ex:
sln(1000; 100; 5)
returns **180**
#end_function

#begin_function
sqrt(x)
Returns the square root of x.
Ex:
sqrt(25)
returns **5**
#end_function

#begin_function
std(args...)
Returns the population standard deviation of a list of values. It measures the amount that the items in the list vary from the average of the list. Since the standard deviation is the square root of the variance, the formula **sqrt(var(args...))** produces the same result as **std(args...)**.
Also see **stds()** and **var()**
Ex:
std(45.7; 32.8; 47.9; 35.3)
returns **6.48281381808856**

std(A1:G12)
returns the population standard deviation of the values in the cell range A1:G12
#end_function

#begin_function
stds(args...)
Returns the sample standard deviation of a list of values. It measures the amount that the items in the list vary from the average of the list. Since the standard deviation is the square root of the variance, the formula **sqrt(vars(args...))** produces the same result as **stds(args...)**.
Also see **std()** and **vars()**
Ex:
stds(45.7; 32.8; 47.9; 35.3)
returns **7.48570860595932**
stds(A1:G12)
returns the sample standard deviation of the values in the cell range A1:G12
#end_function

#begin_function
string(number; [decimals])
Returns **number** as a string with a specified number of decimals.
Ex:
string(1.23456789)
returns **1**
string(1.23456789; 2)
returns **1.23**
#end_function

#begin_function
substring(string; [start]; [n])
Returns **n** characters from string, starting from the character **start**.
Ex:
substring("Hello world!"; 1; 5)
Returns **Hello**

substring("Hello world!"; 7; 5)
Returns **world**
#end_function

#begin_function
sum(args...)
Returns the sum of the list of arguments.
Ex:
sum(12; 13; 17)
returns **42.**
sum(A1:G12)
returns the sum of the values in the cell range A1:G12
#end_function

#begin_function
syd(cost; salvage; life; period)
Calculates the annual depreciation using the "Sum of the Year's Digits" method
The function uses the following formula:

syd = ((cost - salvage) * (life - period + 1)) / (life * ((life + 1) / 2))

Ex:
syd(1000; 100; 5; 2)
returns **240**
#end_function

#begin_function
system(string)
Executes **string** in a UNIX shell. The function always returns *true*.
Also see **eval_in_unix_shell()**
Ex:
system("open /NextApps/Grab.app")
returns **true**
and launches the **Grab** application.

#end_function

#begin_function
tan(x)
Returns the tangent of the angle x. The angle must be expressed in radians.
Ex:
tan(2)
returns **-2.18503986326152**
#end_function

#begin_function
term(payments; interest; future_value)
Calculates the number of periods it will take for equal payments to grow to a specified value, assuming a fixed interest rate.
The function uses the following formula:

term = ln(1 + (future_value * interest / payment)) / ln(1 + interest)

where **ln()** is the natural logarithm
Ex:
term(3600; 0.09; 35000)
returns 7.29
#end_function

#begin_function
textfield_number_value(name)
Returns the value of a text-field as a number. **name** is the number or name of the text-field.
Ex:
textfield_number_value(2)
returns something like **34**
textfield_number_value("my_text")
returns something like **34**
textfield_number_value("my_text")

returns **#invalid_gadget_name**   ; *if my_text does not exist*
#end_function

#begin_function
textfield_string_value(name)
Returns the value of a text-field as a string. **name** is the number or name of the text-field.
Ex:
textfield_string_value(2)
returns something like **a string**
textfield_string_value("my_text")
returns something like **a string**
textfield_string_value("my_text")
returns **#invalid_gadget_name**   ; *if my_text does not exist*
#end_function

#begin_function
time(hour; [minutes]; [seconds])
Returns a time value from **hour**, **minutes** and **seconds**.
Ex:
time(11)
returns **11:00:00**
time(11; 10)
returns **11:10:00**
time(11; 10; 30)
returns **11:10:30**
#end_function

#begin_function
timevalue(string)
Converts a string into a time value.
Ex:
timevalue("18:30")
returns **18:30**

timevalue("18:22:00")
returns **18:22:00**
#end_function

#begin_function
time_add_hours(time; [no_of_hours])
Adds **no_of_hours** to **time.**
Ex:
time_add_hours(11:11:11; 2)
returns **13:11:11**
time_add_hours(23:12:00; 4)
returns **03:12:00**
#end_function

#begin_function
time_add_minutes(time; [no_of_minutes])
Adds **no_of_minutes** to **time.**
Ex:
time_add_minutes(11:11:11; 2)
returns **11:13:11**
time_add_minutes(23:57:00; 4)
returns **00:01:00**
#end_function

#begin_function
time_add_seconds(time; [no_of_seconds])
Adds **no_of_seconds** to **time.**
Ex:
time_add_seconds(11:11:11; 2)
returns **11:11:13**
time_add_seconds(23:59:30; 45)
returns **00:00:15**
#end_function

#begin_function
time_subtract_hours(time; [no_of_hours])
Subtracts **no_of_hours** from **time.**
Note: A time can never be smaller than 00:00:00
Ex:
time_subtract_hours(11:11:11; 2)
returns **09:11:11**
time_subtract_hours(01:20:00; 3)
returns **00:00:00**
#end_function

#begin_function
time_subtract_minutes(time; [no_of_minutes])
Subtracts **no_of_minutes** from **time.**
Note: A time can never be smaller than 00:00:00
Ex:
time_subtract_minutes(11:11:11; 2)
returns **11:09:11**
time_subtract_minutes(00:20:00; 25)
returns **00:00:00**
#end_function

#begin_function
time_subtract_seconds(time; [no_of_seconds])
Subtracts **no_of_seconds** from **time.**
Note: A time can never be smaller than 00:00:00
Ex:
time_subtract_seconds(11:11:11; 2)
returns **11:11:09**
time_subtract_seconds(00:00:40; 50)
returns **00:00:00**
#end_function

#begin_function
today()
Returns a date value for the date of the computer clock.
Ex:
today()
returns something like **26-Feb-93**
date_to_number(today())
returns something like **34025.435937499999**
#end_function

#begin_function
trim(string)
Deletes beginning and ending spaces from **string**. Also, multiple spaces are replaced by one single space.
Ex:
trim("   A   string! ")
returns **A string!**
#end_function

#begin_function
true()
Returns the value *true*.
Also see **false()**
#end_function

#begin_function
upper(string)
Converts all characters in **string** to upper case.
Also see **lower()**
Ex:
upper("hello, HELLO")
returns **HELLO, HELLO**
#end_function

#begin_function
value(string)
Returns the number value of string.
Ex:
value("123.23")
returns **123.23**
value("abc")
returns **0**
#end_function

#begin_function
var(args...)
Returns the population variance of a list of values. It measures the amount that the items in the list vary from the average of the list. Since the standard deviation is the square root of the variance, the formula **std(args...)^2** produces the same result as **var(args...)**.
The population variance is calculated from the following formula:
var = sum((Vi - AVG)^2) / (n - 1)
Vi is the ith item
AVG is the average of the items
n is the number of items.
Also see **std()** and **vars()**
Ex:
var(45.7; 32.8; 47.9; 35.3)
returns **42.02687500000002**
var(A1:G12)
returns the population variance of the values in the cell range A1:G12
#end_function

#begin_function
vars(args...)
Returns the sample variance of a list of values. It measures the amount that the items in the list vary from the average of the list. Since the standard deviation is the square root of the variance, the formula **stds(args...)^2**

produces the same result as **vars(args...)**.
The population variance is calculated from the following formula:
vars = sum((Vi - AVG)^2) / n
Vi is the ith item
AVG is the average of the items
n is the number of items.
Also see **stds()** and **var()**
Ex:
vars(45.7; 32.8; 47.9; 35.3)
returns **56.03583333333336**
vars(A1:G12)
returns the sample variance of the values in the cell range A1:G12
#end_function

#begin_function
wait(milliseconds)
Halts the execution for a number of milliseconds.
Ex:
wait(2000)
will wait for 2 seconds
#end_function

#begin_function
wall_time(expr; [count])
Returns the number of seconds it takes for Questor to execute the **expr** expression **count** number of times.
Note: This function is used for performance tests only.
Ex:
wall_time(wait(2000); 1)
returns something like **2.000216**
wall_time(2 * 2; 100)
returns something like **0.000092**

Note: These times are depending on the load of the computer.
#end_function

#begin_function
week_day(date)
Returns the day of the week for a date.
Also see **year()**, **month()**, **day()** and **year_day()**
Ex:
week_day(12-Aug-1987)
returns **3**
week_day(1992-12-11)
returns **5**
#end_function

#begin_function
window_server_time(type; expr; [count])
Returns the number of seconds it takes for Questor and the Window Server to execute the **expr** expression **count** number of times.
type specifies the type of measurement:
0 - time in the Window Server
1 - time in Questor
2 - % time in the Window Server
Note: This function is used for performance tests only.
Ex:
window_server_time(1; redisplay_windows(); 1)
returns something like **0.728936**

Note: These times are depending on the load of the computer.
#end_function

#begin_function
write(string; [path])
Writes a string into a text file that has been opened with the **open()** function.
It starts writing at the current position of the file pointer.
string is the string that should be written to the file.
path is the file it should be written to.

The file pointer moves to the first character after the inserted string.
If only one file is opened, **path** can be omitted.
If more that one file is opened, **path** can be either the full path name, the full file name or just the file name without extension.
Also see **open()** and **writeln()**
Ex:
open("/tmp/new_file.txt")
Any of these are ok:
write("a string"; "/tmp/new_file.txt")
write("a string"; "new_file.txt")
write("a string"; "new_file")
write("a string")   *; if no other files are open*
close("/tmp/new_file.txt")
#end_function

#begin_function
writeln(string; [path])
Writes a string into a text file that has been opened with the **open()** function.
It starts writing at the current position of the file pointer and adds a carriage return and a line feed after the string.
string is the string that should be written to the file.
path is the file it should be written to.
Also see **open()** and **write()**
Ex:
open("/tmp/new_file.txt")
writeln("a string"; "/tmp/new_file.txt")
close("/tmp/new_file.txt")
#end_function

#begin_function
year(date)
Returns the year for a date.
Also see **month()**, **day()**, **week_day()** and **year_day()**
Ex:
year(12-Aug-1987)

returns **1987**
year(1992-12-11)
returns **1992**
#end_function

#begin_function
year_day(date)
Returns the day of a year for a date.
Also see **year()**, **month()**, **day()** and **week_day()**
Ex:
year_day(12-Aug-1987)
returns **224**
year_day(1992-12-11)
returns **346**
#end_function

#begin_function
year_day_count(date)
Returns the number of days of the year in **date**.
Ex:
year_day_count(92-02-11)
returns **366**
year_day_count(12-Aug-83)
returns **365**
#end_function

#begin_function
year_is_leap(date)
Returns *true* if the year in **date** is a leap year. Otherwise the function returns *false*.
Ex:
year_is_leap(93-12-12)
returns **false**
year_is_leap(12-Aug-92)

```
returns true
#end_function
```