

Lua Scripting

Lua Implementation in SeregonMap

The SeregonMap implementation of Lua is identical to that in other platforms, with the exception of the os library, which is not supported. The online documentation for Lua can be found here:

<http://www.lua.org/manual/5.1/>

Lua is a dynamically typed language. No type definitions are needed for variables in the language; each value carries its own type. Eight default types are used in Lua: nil, Boolean, number, string, userdata, function, thread, and table

Built-in Libraries:

basic functions: supported except for file operations

os:

- not supported: remove, rename, getenv, tmpname, setLocale;
- partial support: os.date these formats not supported: 's', 'u', 'U', 'W', 'w')

string:- fully supported

coroutine: not supported

math: – partial support - no trig or exponent functions

table: – supported

Additional objects provided in MAP:

- map
- mapevent
- mapprojects
- mapapps

Language overview

Global context and objects:

In Lua, all variables are global variables by default. To create local variables, define the variable with the **local** statement:

```
i = 10  <-- global variable
```

```
local i = 1  <-- local variable
```

Lua Examples

```
map.alert(type(a)) -> nil # as 'a' has not been initialized.
```

```
a = 1
```

```
map.alert(type(a)) -> number
```

```
a = false
```

```
map.alert(type(a)) -> Boolean
```

```
a = "a string"
```

```
map.alert(type(a)) -> string
```

```
a = type ← this is assigning a function to a variable.
```

```
map.alert(type(a)) -> function
```

```
a = {}
```

```
map.alert(type(a)) -> table or array
```

```
k = "x"
```

```
a[k] = 10 or a["x"] = 10
```

```
map.alert(a["x"]) -> 10
```

```
a["x"] == a.x
```

```
map.alert(a.x) -> 10
```

```
#a means the last item for array a{}
```

```
For i=1, #a do
```

```
    map.alert(a[i])
```

Basic Lua Expressions

Arithmetic Operators:

+, -, *, /, ^, %, - (negation)

Relational Operators:

<, >, <=, >=, ==, ~=

Logical Operators:

and, or, not

Example 1) when flag is not set, set flag to Aflag value; otherwise, set flag to itself.

```
AFlag = ture
```

```
flag = flag or Aflag < == > if not flag then flag = Aflag end
```

Example 2) select the maximum number from a and b

```
max = (a > b) and a or b
```

String Concatenation Operator:

Example:

```
a = "Hello"
b = a .. "World"
map.alert(b) -> Hello World
```

See the string library for additional string functions.

Escape Operator: “\”

Statements:

If Statements:

```
if Flag == false then
  mapevent.form.showTargetScreen("MyApp","FirstScreen")
else
  mapevent.form.showTargetScreen("MyApp","SecondScreen")
end
```

For Statements:

```
for myrow in mapevent.datatable.filter("MyTable.Id", IdValue, "=")
  do
    if myrow.getColumnValue("MyTable.ColumnA") == ColumnValue then
      myrow.setColumnValue("MyTable.ColumnB", ColumnValue1)
    else
      myrow.setColumnValue("MyTable.ColumnB", ColumnValue2)
    end
  end
end
```

Function Example: (Note: the function has to be defined before it can be used)

```
function recursiveFun (IdValue)
  for myrow in mapevent.datatable.filter("MyTable.Id", IdValue, "=")
    do
      if myrow.getColumnValue("MyTable.ColumnA") == IdValue then
        myrow.setColumnValue("MyTable.ColumnB", IdValue)
      else
        myrow.setColumnValue("MyTable.ColumnB", IdValue+1)
      end
    end
  end
end
if flag == true then
  mapevent.row.setColumnValue("MyTable.ColumnC", 1)
```

```
else
    recursiveFun(2)
end
```

MAP Object Domain

SeregonMap provides in Lua a complete object model of the project, with objects (Lua tables) representing the various objects in a Map project, including application, form, field, grid, gridrow, table, row, field. There are 3 global objects provided by Seregon Map in Lua: map, mapproject and mapapps. These provide access to all aspects of the project. In addition, each event has its own object mapevent which contains data specific to that event, e.g. the current field, the current form, etc.

Map Object Types

SynchResult

Used in the RecordsReceived event after a synchronization has returned new records.

```
Iterator SynchRow newRows()
int newRecordsCount
Iterator String deletedPks()
```

examples

```
for v in mapevent.synchResult.newRecords() do map.alert(v.pkValue) end
for v in mapevent.synchResult.deletedPks() do map.alert(v) end
```

SynchRow

```
String pkValue;
String pkValueSent
getColumnValue(String field) --returns either a String value or a new enumeration of type
NewRecord if the field is a detail table
```

Application

```
Table[“tablename”] tables (indexed by tablename)
Form[“formid”] forms (indexed by form id)
void runSynchRule(String rulename)
```

Form

```
Object [“id”] fields (an array of all fields on this form, including fields on headers, footers and
inside panels, includes both Grids and Fields, ) (indexed by id)
int bgcolor
int forecolor
void close()
void show(int databindtype)   databindtypes: NONE 0; NEW 1; COPY 2; UPDATE 3; FIND 4;
```

selectTab(int tabposition) selects the indicated tab and shows it's form
enableTab(int tabposition, bool enable)

Field

bool visible

int forecolor

int backcolor

Object value returns the contents of this field, normally this is in a string

Grid

GridRow[] rows (indexed by position or pkvalue)

int focusRowIndex

GridRow focusRow

Table table

int rowCount

String focusRowPkValue

filter(String column, String value, String operation)

(filters the rows shown in a grid, operations are =, !=, startswith, contains, endswith, empty, notempty)

Form parentScreen

GridRow getGridRowByPk(String value)

GridRow (a grid row is the visual representation of a TableRow as shown by a Grid)

int forecolor

int backcolor

String pkValue

void delete()

void select()

Object getCellValue(String colname)

void setCellValue(String colname, Object obj)

TableRow tableRow – the table row referenced by this grid row

Table

Row[] rows (array indexed by pkvalue)

int rowCount

TableRow newRow() creates a new row and returns it. The row is not added to the table until
addRow (row) is called

Iterator TableRow all() - (returns all the rows in the table)

IteratorTableRow filter(String field, String findtext, String operation)

- (operations are: =, !=, startswith, contains, endswith, empty, notempty)

(returns a filtered list of the rows in the table)

addRow (TableRow row) adds a new row into the table

example1 (enum all rows)

```
for myrow in mytable.all()
    do print(myrow.getColumnValue("id"));
end;
```

example2 (enum rows where field pk value starts with "aa")

```
for myrow in mytable.filter("pk", "aa", "startswith")
    do print(myrow.getColumnValue ("id"));
end;
```

TableRow

int rowindex

String pkValue (readonly)

Object getColumnValue(String column) returns String or Table if detail field

void setColumnValue(String column, Object value) accepts a String or a Table if the column is a detail field

bool deleted

copy() All fields in the row will be copied

deepCopy(TableRow fieldCopyDefsRow) fieldCopyDefsRow is a row that specifies which fields to copy. Detail tables in the row can also be copied, in which case these row will be given new pk's in the usual way.

Table parentTable

Global Map Objects

mapevent

- contains objects relevant to the event being handled

Field field

Grid grid

Form form

Application app

GridRow gridrow

TableRow row (selected table row loaded to a detail screen with new, copy, update databinding)

String name (name of this event)

bool cancel (valid for beforeshowmenuitems event, if set to true, indicates the menu should not be shown)

bool onload: indicates the reason for a datachanged event

following are for synch events only

Table dataTable

String ruleName

SynchResult synchResult

mapevent object			
Type	Name	Usage	Example
Field	field	mapevent.field	<code>mapevent.field.value = "NewLabel"</code>
	Gets the current focussed field object		
Grid	grid	mapevent.grid	<code>myGridRows = mapevent.grid.gridrows</code> <code>myRowCount = mapevent.grid.rowCount</code> <code>myTable = mapevent.grid.table</code> <code>myFocusRowPkValue = mapevent.grid.focusRowPkValue</code> <code>myFocusRowIndex = mapevent.grid.focusRowIndex</code> <code>myFocusRow = mapevent.grid.focusRow</code> <code>mapevent.grid.filter("ColumnA", "Vlaue")</code> <code>myScreen = mapevent.grid.parentScreen</code>
	Gets current focused grid object		
Form	form	mapevent.form	<code>myScreenFieldList = mapevent.screen.fields</code> <code>myParentGridRow = mapevent.screen.row</code> <code>mapevent.screen.enableTab(tabNum, boolean)</code> <code>mapevent.screen.selectTab(tabNum)</code> <code>mapevent.screen.save()</code> <code>mapevent.screen.close()</code> <code>mapevent.screen.changed(boolean)</code> <code>mapevent.screen.show(int_dataindtype)</code> <code>mapevent.screen.showTargetScreen("appName", "targetScreenName")</code>
	Gets current focused screen object		
Application	app	mapevent.app	<code>myFormsList = mapevent.app.forms</code> <code>myTablesList = mapevent.app.tables</code> <code>mapevent.app.runSynchRule("ruleName")</code>
	Gets the current application instance		
GridRow	gridRow	mapevent.gridRow	<code>mapevent.gridRow.visible = true</code> <code>mapevent.gridRow.forecolor = red</code> <code>mapevent.gridRow.backcolor = black</code> <code>myPkValue = mapevent.gridRow.pkValue</code> <code>mapevent.gridRow.delete(boolean)</code> <code>myValue = mapevent.gridRow.getCellValue("columnName")</code> <code>mapevent.gridRow.setCellValue("columnName", newValue)</code>
	Gets the current focused gridRow object; the columns of gridRow are dependent on how the grid is defined, not dependent on the table that the grid is associated with		

TableRow	row	mapevent.row	<pre> myPkValue = mapevent.row.pkValue mapevent.row.deleted(boolean) myValue = mapevent.row.getColumnValue("ColumnName") mapevent.row.setColumnValue("ColumnName", newValue) mapevent.row.deepCopy(SeregonDataRowOfFieldsNeedCopyList) </pre>
	Gets the current table row object; the columns of row are dependent on how the table is defined, are not depending on the grid		

map

- contains global functions and data

String username

String pin

Row loggedInUserDataRow – the data row returned from the database during login

startGPS()

stopGPS()

appExit()

showProgressScreen(String screenTitle) // wm todo handle b/g thread calls

closeProgressScreen() // WM todo handle b/g thread calls

updateProgress(String txt) // WM todo

chargeCreditCard()

alert(String msg)

String ask(String msg) → returns “OK”, “CANCEL”

error(String) sends log error message to server, displays alert in simulator only

debug(String) sends debug message to monitor log

persist(String name, Object value) stores a value into persistent device storage (String, Number types only)

Object restore(String name)

map object		
Type	Name	Usage
String	username	map.username
String	pin	map.pin
DataRow	loggedInUserDataRow	map.loggedInUserDataRow
Function	startGPS()	map.startGPS()
Function	stopGPS()	map.stopGPS()
Function	appExit()	map.appExit()
Function	showProgressScreen	map.showProgressScreen(“screenTitle”)
Function	closeProgressScreen	map.closeProgressScreen()

Function	updateProgress	map.updateProgress("screenTitle")
Function	chargeCreditCard	map.chargeCreditCard()
Function	alert	map.alert("message")
String	ask	map.ask("message")
Function	error	map.error("message")
Function	debug	map.debug("message")
Function	persist	map.persist(String Name, Object value)
Function	restore	map.restore(String name)

mapproject

Provides access to all forms and data contained within the project through the apps object

Application[] apps (array indexed by application name)

mapproject object			
Application[]	Type	Name	Usage
	List	apps	MyApp = mapproject.apps["myapp"]

mapapps

Provides direct access to all applications within the project

Application[] mapapps (array indexed by application name)

If the application is not yet loaded, then Nil will be returned. An application is loaded when the user starts it from selecting it in the Startscreen or it is started from ShowScreen action.

Application[]	Type	Name	Usage
	List	mapapps	MyApp = mapapps["myapp"]

Examples

Form, Field access (global scope)

```
map.alert ("test1")
myapps = map.apps
myapp = myapps["FMDemo"]
myform = myapp.forms["MainForm"]
myfield = myform.fields["label"]
map.alert ("test2")
map.alert ("screen field value is " . myfield.value)
```

Note an application will return Nil until the user has started it

Table, Row, Field access (global scope)

```
map.alert("test1")
myapps = map.apps
myapp = myapps["FMDemo"]
mytable = myapp.tables["wr"]
myrow = mytable.rows[0]
map.alert("test2")
map.alert("table value is " . myrow.getColumnValue("wr.wr_id"))
```

Field access (event handler scope)

```
map.alert("test1")
map.alert("field value is " . mapevent.field.value)
```

Differences between Bscript and Lua

Global Variables:

In Bscript, the global workspace containing the global variables and functions (as initialized by the GlobalScript) was persisted between the application exits and restart (unless it is retained in memory due to background event handling). Each script ran in a new temporary workspace, but had access to the global workspace.

In Lua, the workspace is not persisted after the application exits. Data can be persisted using the `map.persist()` and `map.restore()` functions. There is one global workspace for each application, which all scripts run in. Any variable not marked as 'local' will be global as per normal Lua conventions. Care should be taken to avoid global variables, especially references to application data such as forms, tables, and form and table data. This may cause memory leaks and out of memory errors. Variable names are case sensitive in Lua.

Background operations.

Supported in Lua, not in BScript

Todo... add AppShutdown method to help persisting of data

Global variables / Lua Execution Context

Each event in the MAP implementation of Lua runs in its own state (similar to a Lua coroutine) which is initially a copy of the GlobalScript state. Any variables and functions defined in the GlobalScript are available to any event. Variables and functions declared in an event handler are local to that event handler only, even if not defined as local.

Persisting Variables

Variables specific to an application can be persisted in a table defined for that purpose, a RowObject table containing a column for each variable to be persisted is a good solution. The values can then be set and retrieved in this way:

setting a variable: `persistTable.rows[0].setColumnValue("var1", newval)`

retrieving the value of a variable: `val = persistTable.rows[0].getColumnValue("var1")`

Variables that are global to all applications, but user-specific, can be stored in a similar way in columns defined in the table used for logging in users (if defined).

Variables that are global and available to all applications regardless of context can be stored in the `map.persist()` and `map.restore()` functions.

Data stored using the above methods will be stored in the device flash memory and will retain their value even if the application is closed and restarted.

Threading

Lua scripts executed as part of an action can optionally run in the background, with or without a built-in progress dialog while the script is running. The progress dialog shows until the script completes, or it is closed by with the `CloseProgressScreen` function. If the script closes the progress screen, the script will continue to run but the user will then be able to continue to interact with the program while the script is running. Care should be taken when interacting with screens and data after this point, since the user may open or close screens or reload them with new data. Lua scripts are always initiated from the main thread by default, except for the `RecordsReceived` script, which is always in a background thread.

Example wait script:

```
a = 1
while a < 300000 do  a = a + 1 end
```

This script executes in about 25 seconds on a simulator.

Concurrent thread operations

Re-entrancy

Some script actions may themselves cause a script to run while the original script is still executing, e.g. a `ShowScreen` action with a form that itself has a script in its `OnLoad` action list. At present Lua cannot run such re-entrant scripts.

Optimizing your scripts

Lua is an interpreted language, it does not have an optimizer. Care should be taken to avoid duplication of code. For example, in the following code, the lua interpreter has to look up the required table and row twice:

```
myfield1_val = table["mytable"].rows["myrowpk"].getColumnValue("myfield1")
myfield2_val = table["mytable"].rows["myrowpk"].getColumnValue("myfield1")
```

To optimize this, use an intermediate variable to hold the required row:

```
myrow = table["mytable"].rows["myrowpk"]  
myfield1_val = myrow.getFieldValue("myfield1")  
myfield2_val = myrow.getFieldValue("myfield1")
```

This is especially important in loops where a property or method access may occur many times.

Using the Lua Debugger

Map Studio 3.1 contains a built-in script debugger allowing breakpoints, and inspection of variables and the execution stack.

To enable script debugging, run both the simulator and Map Studio on one machine, and publish your project using the 'Debug' button. In the Script debugger wizard page, check 'Enable Script Debugging', and select the scripts you would like to debug.

Publish the project in the normal way, after publishing the script editor will be displayed, with the title 'Waiting for connection'. Open any scripts you wish to debug and set breakpoints as required, and then start your application. As soon as a script needs to execute, the application will show a dialog prompting you that it is connecting to the debugger. The script editor will show 'Connected – inactive' showing the device has connected but no script is running. This is the last chance to set any breakpoints you need in a global script or onload actions for the initial screen. Once the script has started to run, the script editor will show 'Connected – waiting for device'. When a breakpoint has been hit and script execution has stopped, the script editor will show the current location and the local and global variables. If the script stalls on the device, for example, if script is showing an alert, the script editor will show 'Waiting for device' until control returns to the debugger. Once the script has completed execution, the debugger will return to 'inactive' state.