



Seregon Mobile Application Platform

Lua Scripting

Version 3.2

Table of Contents

Lua Implementation in SeregonMap.....	1
Language overview.....	1
Global context and objects:.....	1
Lua Examples.....	2
Basic Lua Expressions.....	2
Statements:.....	3
MAP Object Model.....	5
Map Object Types.....	5
Application.....	5
Color.....	5
Field.....	5
FieldAction.....	5
Form.....	6
Grid.....	6
GridRow (a grid row is the visual representation of a TableRow as shown by a Grid).....	7
SynchResult.....	7
SynchRow - Depreciated.....	7
Table.....	7
TableRow.....	8
Global Map Objects.....	8
map.....	8
mapapps.....	9
mapcolor.....	10
mapevent.....	10
mapproject.....	12
Examples.....	12
Differences between BScript and Lua.....	13
Global Variables:.....	13
Background operations.....	13
Global variables / Lua Execution Context.....	13
Persisting Variables.....	13
Threading.....	13
Concurrent thread operations.....	14
Re-entrancy.....	14
Showing an update screen based on a row.....	14
Optimizing your scripts.....	14
Using the Lua Debugger.....	14
Firewall Setup.....	15
Using the Debugger.....	15

Lua Implementation in SeregonMap

The SeregonMap implementation of Lua is identical to that in other platforms, with the exception that some of the standard libraries are not completely 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 – the following 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 – global functions and properties
- mapevent – properties relating to the current event
- mapproject – the top-level entity in the object model.
- mapapps – a shortcut to access the application list in a project
- mapcolor – a list of colors used in fields – matches the standard colors provided by Web pages

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])
end
```

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
```

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
if flag == true then
    mapevent.row.setColumnValue("MyTable.ColumnC", 1)
else
```

```
recursiveFun(2)  
end
```

MAP Object Model

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

Application

Table[“tablename”] **tables** (indexed by tablename)
Form[“formid”] **forms** (indexed by form id)
void **runSynchRule**(String rulename)
void **runSynchRuleAfterDelay** (String rulename, int delayInSeconds)
Add the rule to the queue. Rule will not execute until at after the delay has passed

Color

This is a list of available colors, using standard web color naming, for example:
mapcolor.MEDIUMTURQUOISE
mapcolor.WHITE

Field

bool **visible**
int **forecolor** colors are standard RGB or use the mapcolor constants
int **backcolor** colors are standard RGB or use the mapcolor constants
Object **value** returns the contents of this field, normally this is in a string
void **action**(FieldAction) performs a built-in field action – see FieldAction object
String **validation** When set, prevents the screen from being saved.

FieldAction

This is a list of available colors, using standard web color naming, for example:
myfield.action(mapfieldaction.Image_View)

Available field actions:

Phone_Dial
Email_SendEmail
GPS_ShowLocation
GPS_GetLocation
Image_View

Image_Camera
Image_FromFile
Media_Record
Media_Play
Media_Append

Form

Object [“id”] **fields** Indexed by id or index
(an array of all fields on this form, including fields on headers, footers and inside panels, includes both Grids and Fields,)

int **backcolor** Colors are standard RGB or use the mapcolor constants

int **forecolor** Colors are standard RGB or use the mapcolor constants

void **close**()

void **show**(int databindtype) databindtypes: *NONE* 0;*NEW* 1;*COPY* 2;*UPDATE* 3

If UPDATE or COPY is selected, the currently selected row in the parent grid is used for the databind operation. Use grid.focusRow to set the selected row first if required

selectTab(int tabindex) Selects the indicated tab and shows its form (index is 0-based)

enableTab(int tabposition, bool enable)
Sets whether or not a tab is usable

bool **changed** Get or set whether or not there has been a change. For menu visibility.

void **save**() Saves the current form

Grid

GridRow[] **gridrows** (indexed by position or pkvalue)

int **focusRowIndex** Sets the focus row by index or returns the index of the current focus row

GridRow **focusRow** Can be set with either a GridRow or TableRow

TableRow **row** Returns the parentRow for the form of the current grid.

Table **table**

int **rowCount**

String **focusRowPkValue** Sets the focus row by pk value, or returns the pk value of the current focus row.

filter(String column, String value, String operation)
(operations are =, !=, startswith, contains, endswith, empty, notempty)
Filters the rows shown in a grid

sort(String column) (toggle sort direction if already sorted)

sortAsc(String column)

sortDesc(String column)

void **showDetailForm**(String detailFormName, int databindtype)

Show detail form associated with this grid, see Form for databind codes

Form **parentScreen**

GridRow **getGridRowByPk**(String value)

void **startCamera**(String cameraField)

Starts the add photo process for the grid. CameraField is a blob field in the data table for the grid. Will allow continuous addition of pictures to the grid.

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

int **forecolor** Colors are standard RGB or use the mapcolor constants

int **backcolor** Colors are standard RGB or use the mapcolor constants

String **pkValue**

void **delete**()

void **select**(int index) Sets the focus row to the indicated index

Object **getCellValue**(String colname)

void **setCellValue**(String colname, Object obj)

TableRow **tableRow** The table row referenced by this grid row

SynchResult

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

Iterator String **newRows**() List of newly arrived pks

int **newRecordsCount**

Iterator String **deletedPks**() List of pks marked for removal

examples

```
for recpk in mapevent.synchResult.newRecords() do map.alert(recpk) end
```

```
for recpk in mapevent.synchResult.deletedPks() do map.alert(recpk) end
```

SynchRow - Deprecated

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

Table

Row[] **rows** Array indexed by pkvalue or index

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
 Iterator **TableRow** **filter**(String field, String findtext, String operation)
 - (operations are: =, !=, startswith, contains, endswith, empty, notempty)
 Returns a filtered list of the rows in the table
 void **clear**() Removes all rows from the data 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**

Example:

```
app.table["tablename"].rows["pk"].deleted = true
```

Global Map Objects

map

- contains global functions and data

map object		
Type	Name	Usage

String	username	map.username
String	Pin	map.pin
DataRow	loggedInUserDataRow	map.loggedInUserDataRow
	stores the data row returned from the database during login	
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")
	returns "OK", "CANCEL"	
Function	error()	map.error("message")
	Logs an error to the logging system	
Function	debug()	map.debug("message")
	Logs a message to the logging system	
Function	persist()	map.persist(String Name, Object value)
	stores a value into persistent device storage	
Function	restore()	map.restore(String name)
Function	showHtml()	map.showHtml(String html)
Show the html data in a new full screen		
Function	showImage()	map.showImage(CellContents)
Function	Show the image (obtained from a table or grid cell) in a new full screen	
	showImages()	map.showImage(Table, ColumnName)
Show the images from the given table in a new full screen. One image is shown at a time. Menu options provides navigation between images. Images are obtained from the given ColumnName for all rows in the table.		
Function	int yesnodialog(String text)	res = map.yesnodialog("question")
	Shows a dialog and returns a string, "YES", "NO", "CANCEL"	

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.

mapcolor

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

mapcolor object			
	Type	Name	Usage
Color	Constant	color	Color.WHITE
Returns a standard web color for use in forecolor, backcolor properties			

mapevent

- contains objects relevant to the event being handled

mapevent object			
Type	Name		Example
Field	field		<i>mapevent.field.value = "NewLabel"</i> <i>mapevent.field.forecolor = mapcolor.red</i>
		Gets the current focused field object	
Grid	grid		<i>myGridRows = mapevent.grid.gridrows</i> <i>myRowCount = mapevent.grid.rowCount</i> <i>myTable = mapevent.grid.table</i> <i>myFocusRowPkValue = mapevent.grid.focusRowPkValue</i> <i>myFocusRowIndex = mapevent.grid.focusRowIndex</i> <i>myFocusRow = mapevent.grid.focusRow</i> <i>mapevent.grid.filter("ColumnA", "Value")</i> <i>myScreen = mapevent.grid.parentScreen</i>
		Gets current focused grid object	
Form	form		<i>myScreenFieldList = mapevent.form.fields</i> <i>myParentGridRow = mapevent.form.row</i> <i>mapevent.form.enableTab(tabNum, boolean)</i> <i>mapevent.form.selectTab(tabNum)</i> <i>mapevent.form.save()</i> <i>mapevent.form.close()</i> <i>mapevent.form.changed(boolean)</i> <i>mapevent.form.show(int_dataabindtype)</i> <i>mapevent.form.showTargetScreen("appName", "targetScreenName")</i>
		Gets current focused screen object	
Application	app		<i>myFormsList = mapevent.app.forms</i> <i>myTablesList = mapevent.app.tables</i> <i>mapevent.app.runSynchRule("ruleName")</i>
		Gets the current application instance	

GridRow	gridrow	<pre> mapevent.gridRow.visible = true mapevent.gridRow.forecolor = mapcolor.red mapevent.gridRow.backcolor = mapcolor.black myPkValue = mapevent.gridRow.pkValue mapevent.gridRow.delete(boolean) myValue = mapevent.gridRow.getCellValue("columnName") mapevent.gridRow.setCellValue("columnName", newValue) </pre>
	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	<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 assigned to the current screen by the databinding type;	
bool	cancel	<code>mapevent.cancel = true</code>
	valid for the beforeshowmenu event, if set to true, indicates the menu should not be shown	
bool	onload	
	indicates the reason for a DataChanged event is that the form is being loaded for the first time (rather than as a result of a synch operation).	
Table	dataTable	
	for Synch Completed event only: the table for which this event occurred	
String	ruleName	
	for Synch Completed event only: the name of the rule that was run	
SynchResult	synchResult	
	for Synch Completed event only: an object containing the records updated and deleted	
	print()	<code>mapevent.print("
")</code>
	For Browser html scripts, prints text to the html page	

mapproject

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

mapproject object			
	Type	Name	Usage
Application[]	List	apps[]	MyApp = mapproject.apps["myapp"]
			array indexed by application name

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 same workspace is used in each application. 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.

Not supported at present.

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

Global variables / Lua Execution Context

Each event in the MAP implementation of Lua runs in a shared state, so global variables declared in one script are available to subsequent scripts. Variables should be declared local if possible.

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.

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. Lua is fully re-entrant. Note if a ShowScreen is executed after a progress screen has been shown with ShowProgress, the Progress Screen will be automatically closed before the new screen is shown.

Showing an update screen based on a row

If you want to show the details of a specific row, for example if the Synch Completed event indicates that a record has been updated, and you would like to automatically show that record's details on a form. To do this, use a script that selects the row in the parent grid, and then execute a ShowScreen function using a datbind type of update. This will show the form using the currently selected grid row.

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

Firewall Setup

The Lua debugger connects to Map Studio using an TCP port. In order to make the connection. Port 9088 must be opened on your Windows Firewall, if enabled.

Using the Debugger

The simulator and Map Studio must be running on the same machine, and publish your project using the 'Debug' button. In the Script debugger wizard page, check 'Enable Script Debugging',

Check the scripts you would like to debug.

After publishing the script editor will be displayed with the title 'Debugger waiting for connection'.

Open any scripts you wish to debug, set breakpoints as required, and then start your application on the device. When the script starts to execute, the device will show a dialog “Connected to debugger. Press Ok when ready to continue”. The script editor will show the title 'Connected – inactive' showing the device has connected but no script is running. This is the last chance to set any breakpoints you need before running the script.

You can set a breakpoint in any script by selecting the line and then using the Debug/Toggle Breakpoint menu command or the shortcut key F9. A red dot should appear in the margin. Do not click in the left hand margin – this only sets a bookmark.

When ready, press 'Ok' on the simulator.

The script will start to run and the script editor will then show 'Debugging – Script running' until a breakpoint is hit..

When a breakpoint has been hit, script execution will stop, the following state will occur:

- The script editor will show 'Connected – Code execution paused’

- The device will show a Dialog ' Script: Stopped by debugger',

- The script editor will open the currently executing script in a window, showing the current location. The local and global variables and call stack tabs will populate.

- Pressing F5 to resume execution of the script will return the state to 'Debugging – Script running' until the next breakpoint is reached, or the script terminates, at which point the state returns to 'Connected – inactive'

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 the 'Connected - inactive' state.