digitalSTROM Server Scripting

digitalSTROM

Version: tag: v1.1-branch[*]

June 19, 2014

[*]Revision: b9175b4666c1b95435a7813d996eb6d96cad7ddc

Contents

1 Introduction

1.1 Scripting inside the dSS

The dSS has the ability to run scripts inside it using the JavaScript interpreter SpiderMonkey. While it doesn't support running a script using a dedicated api, scripts can be run as a result to an "event".

The binding between events and scripts is either statically configured in subscription xml files or dynamically done at runtime using the JSON subscription API methods.

The following example shows how to run a script *data/initialize.js* on system startup triggered by the event *model_ready*.

Listing 1: Script Example

```
<subscription event-name="model_ready" handler-name="javascript">
  <parameter>
    <parameter name="filename1">data/initialize.js</parameter>
  </parameter>
</subscription>
```

1.2 Event handlers

If a script gets invoked by a event handler the global variable *raisedEvent* is supplied to the script runtime environment and forwards context information about the event and the subscription to the event handler script.

For example the raisedEvent for a subscription to a *callScene* event has the following members:

Listing 2: Variable raisedEvent

```
raisedEvent.name = callScene
raisedEvent.source = [object Object]
  raisedEvent.source.set = .zone(4011).group(1)
  raisedEvent.source.groupID = 1
  raisedEvent.source.zoneID = 4011
  raisedEvent.source.isApartment = false
  raisedEvent.source.isGroup = true
  raisedEvent.source.isDevice = false
raisedEvent.parameter = [object Object]
  raisedEvent.parameter.groupID = 1
  raisedEvent.parameter.sceneID = 32
  raisedEvent.parameter.zoneID = 4011
  raisedEvent.parameter.originDeviceID = 3504175fe0000000000183f2
raisedEvent.subscription = [object Object]
  raisedEvent.subscription.name = callScene
```

1.3 Event Documentation

Details about the digitalSTROM Server events and subscriptions can be found in the system-interfaces documentation.

1.4 Exceptions

The SpiderMonkey Engine supports exception handling which should be used when possible, otherwise the script execution may stop unexpectedly. Javascript Exceptions are logged into the dss logfile.

1.5 Environment

The dSS uses a collaborative scripting environment which relies on each script running as fast as possible to completion. There is only a single scripting instance which executes only one subscription context at a time. Therefore one blocking script may affect all other subscriptions and execution of other script handlers.

2 Global Functions

2.1 Print

Listing 3: print

```
print(arg1, ...)
```

Prints variable arguments to the dSS logfile. The log entries are prefixed by "JS:".

2.2 Timeout

Listing 4: setTimeout

```
setTimeout(func, timeoutMS)
```

Calls the function *func* after at least *timeoutMS* milliseconds have elapsed. Note that the script will need to be run completely for the callback to happen.

Notice The timeout callback function is scheduled on a new thread and keeps the javascript interpreter context in memory. Please keep in mind that system ressources may be limited and use setTimeout only for selected and short timeout periods.

2.3 Logger

This class provides a convenient way to write log messages to different files. The logfiles are written to the dSS Javascript log directory.

Listing 5: Logger

```
// constructor
var L = new Logger(logFileName);
```

Creates a new log file.

Listing 6: logln

```
L.logln(message)
```

Writes the message to the logfile.

3 Events

Event classes allow to create and send new events to the dSS. To execute an Event in a later point in time there are two ways with different event subclasses, the TimedEvent and TimedICalEvent.

3.1 Event

3.1.1 Constructor

```
var E = new Event(name [, parameters])
```

The name is mandantory and identifies the event. The parameter list is optional.

3.1.2 raise

```
E.raise()
```

Raises the event and actually appends it to the dSS event queue.

3.1.3 Example

```
var evt = new Event("testevent", {
     zoneID: raisedEvent.source.zoneID,
     groupID: raisedEvent.source.groupID,
     sceneID: raisedEvent.parameter.sceneID
});
evt.raise();
```

3.2 TimedEvent

To execute the Event in a later point in time the TimedEvent class has an additional time parameter. For example the string "+10" will raise the event after 10 seconds.

The event object is stored in the property tree, where it can be cancelled by deleting the node represting the TimedEvent. The node is stored in */system/EventInterpreter/ScheduledEvents/ID*. A TimedEvent returns its ID when the raise() function is executed.

3.2.1 Constructor

```
// constructor
var TE = new TimedEvent(name, time [, parameters])
```

The parameters name and time are mandatory. The time argument is a relative time duration in seconds and must be given as a string (e.g. "+5"). The event is executed after this duration.

3.2.2 raise

```
var id = TE.raise()
```

Raises the event and returns the ID.

3.3 TimedICalEvent

To execute an event periodically or at defined date and time the *TimedICalEvent* uses an iCal recurrence rule and an iCal start time according to "RFC 2445" (http://www.ietf.org/rfc/rfc2445.txt).

The event object is stored in the property tree, where it can be cancelled by deleting the node representing the TimedICalEvent. The node is stored in */system/EventInterpreter/ScheduledEvents/ID*. A TimedICalEvent returns its ID when the raise() function is executed.

3.3.1 Constructor

```
// constructor
var TIcal = new TimedICalEvent(name, iCalStartTime, iCalRRule [,
    parameters])
```

Name, iCalStartTime and iCalRRule are mandatory. Optionally parameters may be passed along with the event.

3.3.2 raise

```
var id = TIcal.raise()
```

Raises the event and returns the ID.

3.3.3 Example

```
var timerEvent = new TimedICalEvent("timer", "20120101T161500", "FREQ=
    WEEKLY;BYDAY=MO,TU,WE,TH,FR", {
   timername: 'Test1',
   action_type: 'execute'
});
var timedID = timerEvent.raise();
```

4 Apartment

4.1 Apartment Static Functions

Apartment and data model functions in the global name space.

4.1.1 getName

```
var string = getName()
```

Returns the name of the apartment.

4.1.2 setName

```
setName(name)
```

Sets the name of the apartment.

4.1.3 getDevices

```
var Set = getDevices()
```

Returns a Set that contains all devices of the apartment.

4.1.4 getZones

```
var array = getZones();
```

Returns an array that contains all zones of the apartment.

4.1.5 getZoneByID

```
var Zone = getZoneByID(ZoneID)
```

Returns the zone object with the given numerical id. Returns null if not found.

4.1.6 getDSMeters

```
var array = getDSMeters();
```

Returns an array that contains all dSMeters of the apartment.

4.1.7 getDSMeterByDSID

```
var dSMeter = getDSMeterByDSID(MeterDSID)*
```

Returns the dSMeter object with the given dSID. Returns null if not found.

4.1.8 getConsumption

```
var integer = getConsumption()
```

Returns the current power consumption of the installation, which is calculated as the sum of all dSMeter power measurements.

4.1.9 getEnergyMeterValue

```
var integer = getEnergyMeterValue()
```

Returns the energy meter value of the installation, which is calculated as the sum of all dSMeter energy meter values.

4.1.10 getState

```
var State = getState(StateName);
```

Returns State object with the given name. Returns null if not found.

4.1.11 registerState

```
registerState(StateName, isPersistent);
```

Registers a new state with the given name. The boolean isPersistent flag defines if the state should be saved and restored to the last known value when the dSS is restarted.

4.2 Apartment Global Constants

A few data model constants are available in the global name space.

```
Scene.Off = 0
Scene.User1 = 5
Scene.User2 = 17
Scene.User3 = 18
Scene.User4 = 19
Scene.Dec = 11
Scene.Inc = 12
Scene.Min = 13
Scene.Max = 14
Scene.Stop = 15
Scene.DeepOff = 68
Scene.StandBy = 67
Scene.Bell = 73
Scene.Panic = 65
Scene.Alarm = 74
Scene.Present = 71
Scene.Absent = 72
Scene.Sleeping = 69
Scene.WakeUp = 70
Scene.RoomActivate = 75
Scene.Fire = 76
Scene.Smoke = 77
Scene.Water = 78
Scene.Gas = 79
Scene.Bell2 = 80
Scene.Bell3 = 81
Scene.Bell4 = 82
Scene.Alarm2 = 83
Scene.Alarm3 = 84
Scene.Alarm4 = 85
Scene.WindActive = 86
Scene.WindInactive = 87
Scene.RainActive = 88
Scene.RainInactive = 89
Scene.HailActive = 90
Scene.HailInactive = 91
```

5 Devices

Device objects can be generated by the getting a set of devices using Apartment.getDevices() or Apartment.getZoneByID(zoneId).getDevices() and then filtering the set with the methods byName() or byDSID().

Listing 7: Example

```
var allDevs = getZoneByID(1234).getDevices();
var D = allDevs.byName("Stehlampe");
```

5.1 Device Properties

The following properties are available on device objects. These properties are read-only.

Property	Type	Description
className	string	=== "Device"
dsid	string	the unique dSID of this device
name	string	the user name for this device
zoneID	int	the zoneID in which the device is
circuitID	int	the dSID of the dSMeter to which the device is attached
functionID	int	the system defined functionality bitfield
revisionID	int	the firmware revision
productID	int	the product revision
isPresent	int	the present flag signaling if a device is currently accessible
lastCalledScene	int	the last scene command that was sent to this device

5.2 Device Methods

A device object supports the following methods. These methods have a corresponding JSON API call.

Notice Some methods wait for device response and block the scripting environment until the operations is complete.

5.2.1 callScene

```
D.callScene(sceneNr [, SceneAccessCategory])
```

Executes the scene call with the given sceneNumber. The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.2 saveScene

```
D.saveScene(sceneNr)
```

5.2.3 undoScene

```
D.undoScene(sceneNr [, SceneAccessCategory])
```

The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.4 nextScene

```
D.nextScene([SceneAccessCategory])
```

The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.5 previousScene

```
D.previousScene([SceneAccessCategory])
```

The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.6 turnOn

```
D.turnOn([SceneAccessCategory])
```

The turnOn() method effectively is the same then executing callScene(Scene.Max). The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.7 turnOff

```
D.turnOff([SceneAccessCategory])
```

The turnOff() method effectively is the same then executing callScene(Scene.Min). The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.8 blink

```
D.blink([SceneAccessCategory])
```

The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.9 setValue

```
D.setValue(value [, SceneAccessCategory])
```

The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.10 increaseValue

```
D.increaseValue([SceneAccessCategory])
```

The increaseValue() method effectively is the same then executing callScene(Scene.Inc). The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.11 decreaseValue

```
D.decreaseValue([SceneAccessCategory])
```

The decreaseValue() method effectively is the same then executing callScene(Scene.Dec). The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

5.2.12 getConfig

```
var num = D.getConfig(configClass, configIndex)
```

5.2.13 getConfigWord

```
var num = D.getConfigWord(configClass, configIndex)
```

5.2.14 setConfig

```
D.setConfig(configClass, configIndex, configValue)
```

5.2.15 getOutputValue

```
var num = D.getOutputValue(outvalIndex)
```

5.2.16 setOutputValue

```
D.setOutputValue(outvalIndex, value)
```

5.2.17 getSensorValue

```
var num = D.getSensorValue(sensorIndex)
```

5.2.18 getSensorType

```
var num = D.getSensorType(sensorIndex)
```

5.2.19 getPropertyNode

```
var prop = D.getPropertyNode()
```

Returns the property tree object of this device.

6 Sets

Set objects are a container of devices, either by the getting a set of devices using Apartment.getDevices() or Apartment.getZoneByID(zoneId).getDevices() or by manually combining device objects using the set object methods.

Listing 8: Example

```
var S = getDevices();
```

6.1 Set Properties

The following properties are available on set objects. These properties are read-only.

Property	Type	Description
className	string	=== "Set"

6.2 Set Methods

A set object supports the following methods.

6.2.1 perform

```
S.perform(function)
```

Calls *function* with a device object parameter for every device contained in the set.

Listing 9: Example

```
S.perform(function(device) { print("Device name:" + device.name) })
```

6.2.2 length

```
var num = S.length()
```

6.2.3 combine

```
var set = S1.combine(S2)
```

6.2.4 remove

```
var set = S.remove(device)
```

6.2.5 byName

```
var device = S.byName("Klingel")
```

6.2.6 byDSID

```
var device = S.byDSID("3504175fe0000000000182f6")
```

6.2.7 byFunctionID

```
var set = S.byFunctionID(1020)
```

6.2.8 byZone

```
var set = S.byZone(12345)
```

6.2.9 byDSMeter

```
var set = S.byDSMeter("3504175fe0000010000004d9")
```

6.2.10 byGroup

```
var set = S.byZone(2)
```

6.2.11 byPresence

```
var set = S.byPresence(1)
```

6.2.12 byTag

```
var set = S.byTag("abcd")
```

7 Zone

Zone objects represent a zone of the installation. The zone object can be generated by the global function getZones() or getZoneByID().

Listing 10: Example

```
var arrZone = getZones()
var Z = arrZone[0]
```

7.1 Zone Properties

The following properties are available on zone objects. These properties are read-only.

Property	Type	Description
className	string	=== "Zone"
id	string	the numerical id of this zone
name	string	the user defined name for this zone
present	int	the present flag signaling if the zone is currently accessible

7.2 Zone Methods

A Zone object supports the following methods.

7.2.1 getDevices

```
var array = Z.getDevices()
```

Returns the list of device objects in this zone.

7.2.2 callScene

```
Z.callScene(groupID, sceneID [, forceFlag, SceneAccessCategory])
```

Executes the scene call in the zone according to group and scene id. The optional forceFlag set to 'true' enables local priority override. The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

7.2.3 undoScene

```
Z.callScene(groupID, sceneID [, SceneAccessCategory])
```

Undoes the scene call in the zone according to group and scene id. The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

7.2.4 blink

```
Z.blink([SceneAccessCategory])
```

The optional parameter SceneAccessCategory is a string where supported values are:

SceneAccessCategory	Description
"manual"	call issued directly by the user
"timer"	call issued as a result of a timed event
"algorithm:"	call issued by an app as a result of some automated action

7.2.5 getPowerConsumption

```
var num = Z.getPowerConsumption()
```

Returns the current consumption as sum of all active devices in this zone.

Notice This will always return 0 as long as single device consumption values are not available.

7.2.6 pushSensorValue

```
Z.pushSensorValue(SourceDSID, sensorType, sensorValue)
```

7.2.7 getPropertyNode

```
var prop = Z.getPropertyNode()
```

Returns the property tree object of this zone.

8 dSMeter

dSMeter objects represent a digitalSTROM Meter of the installation. The dSMeter object can be generated by the global function getDSMeterByDSID(), or by getting the array of all digitalSTROM Meters and

Listing 11: Example

```
var M = getDSMeterByDSID("3504175fe0000010000004d9")
```

8.1 dSMeter Properties

The following properties are available on dSMeter objects. These properties are read-only.

Property	Type	Description
className	string	=== "DSMeter"
dsid	string	the unique dSID of this digitalSTROM Meter
name	string	the user defined name for this digitalSTROM Meter
present	int	the present flag signalling if the device is currently accessible

8.2 dSMeter Methods

A dSMeter object supports the following methods.

8.2.1 getPowerConsumption

```
var num = M.getPowerConsumption()
```

Returns the current power consumption in [W] of this circuit.

8.2.2 getEnergyMeterValue

```
var num = M.getEnergyMeterValue()
```

Returns the accumulated energy meter value in [Ws] for this circuit.

8.2.3 getCachedPowerConsumption

```
var num = M.getCachedPowerConsumption()
```

Returns the last value that was returned by the device. This method does not issue a new request to the device.

8.2.4 getCachedEnergyMeterValue

```
var num = M.getCachedEnergyMeterValue()
```

Returns the last value that was returned by the device. This method does not issue a new request to the device.

8.2.5 getPropertyNode

```
var prop = M.getPropertyNode()
```

Returns the property tree object of this dSMeter.

9 Metering

This class allows to access the metering data of the digitalSTROM Meters.

9.1 Metering Static Methods

9.1.1 getResolutions

```
var array = Metering.getResolutions()
```

Returns an array with the available sample rates and different series. The *resolution* field gives the interval time in seconds, the *type* field is either "energy", "energyDelta" or "consumption".

9.1.2 getSeries

```
var array = Metering.getSeries()
```

Returns the stored series.

9.1.3 getAggregatedValues

```
var array = Metering.getAggregatedValues(dsid, type, resolution, unit,
    startTime, endTime, numValues)
```

Returns an array with the stored metering values for digitalSTROM Meter selected by the dsid string, each parameter selecting a subset of the metering series:

- possible type values are "energy", "energyDelta" or "consumption"
- the unit is either "Wh" (default) or "Ws"
- the startTime and endTime allow to define a window with unix timestamps (both may be 0 to effectively disable the start or endtime function)
- the numValues parameters give the maximum number of requested metering samples

10 Property

This class allows to access the property tree structure and the datamodel of the dSS.

10.1 Property Static Functions

10.1.1 setProperty

```
Property.setProperty(propPath, value)
```

Sets the property at *propPath* to *value*. If the property key already exists the type of the value has to match the existing property value type.

10.1.2 getProperty

```
Property.getProperty(propPath)
```

Returns the value at *propPath*.

Listing 12: Example

```
var t = Property.getProperty("/system/uptime")
print("dss uptime", t)
```

10.1.3 setListener

```
var listenerID = Property.setListener(propPath, func)
```

Sets up a listener callback and calls *func* on any change to *propPath* or its children. Returns an ID to be used for removal of the listener.

10.1.4 removeListener

```
Property.removeListener(listenerID)
```

Removes a previously installed listener.

10.1.5 hasFlag

```
var bool = Property.hasFlag(propPath, flagName)
```

Returns true if the given flag is set.

10.1.6 setFlag

```
Property.setFlag(propPath, flagName, value)
```

Sets the boolean *value* of the *flagName*.

10.1.7 getNode

```
var prop = Property.getNode(path)
```

Returns a property object at the given *path* location.

10.1.8 store

```
Property.store()
```

Persists the properties of the private subtree of the script.

10.1.9 load

```
Property.load()
```

Loads previously persisted properties.

10.2 Property Methods

A property object can be obtained with the getPropertyNode() of Device, DSMeter or Zone objects. For access to any property tree node use the static method Property.getNode(path).

A property object supports the following methods.

10.2.1 getValue

```
var value = P.getValue()
```

Returns the value of the property. The type of *value* depends on the type of the property value, which can be either a string, a numerical or a boolean value.

10.2.2 setValue

```
P.setValue(value)
```

Sets the value of the property to *value*.

10.2.3 setListener

```
var listenerID = P.setListener(func)
```

Registers a callback that gets called if the property tree object or its subnodes change.

10.2.4 removeListener

```
P.removeListener(listenerID)
```

Removes a previously registered callback.

10.2.5 getChild

```
var prop = P.getChild(relativePath)
```

Returns a property object child-node at the relative path or *null* if it doesn't exist

10.2.6 getChildren

```
var array = P.getChildren()
```

Returns an array of children, an empty one if none present.

10.2.7 getParent

```
var prop = P.getParent()
```

Returns the parent property object.

10.2.8 removeChild

```
var bool = P.removeChild([nodeObj|string])
```

Returns *true* if the child has been removed. Para

10.2.9 hasFlag

```
var bool = P.hasFlag(flagName)
```

Returns true if the given flag is set.

10.2.10 setFlag

```
P.setFlag(flagName, value)
```

Sets the boolean *value* of the flag.

10.2.11 getName

```
var string = P.getName()
```

Returns the name of the property object.

10.2.12 getPath

```
var string = P.getPath()
```

Returns the full path of the property object.

11 Network Connectivity

The scripting classes easycurl and easylist allow to use the easycurl interface from "libcurl" (http://curl.haxx.se/libcurl/). Supported protocols are HTTP and HTTPS.

Notice The perform() functions blocks the scripting environment until the operations is complete.

Notice cURL wrapper class for scripting is available since dSS release 1.5.0.

11.1 easycurl

The easycurl class performs the network operations. Example "scripts" https://gitorious.digitalstrom.org/dss/dss-mainline/blobs/master/examples/scripts/easytest.js and a http wrapper can be found in the source code folder ./examples/scripts/.

11.1.1 Constructor

```
var C = new easyscurl()
```

11.1.2 setopt

```
C.setopt(C.option, value)
```

Set a libcurl option, for CURLopt reference see http://curl.haxx.se/libcurl/c/curl_easy_setopt.html.

Listing 13: Example

```
C.setopt(C.CURLOPT_HTTPGET, 1);
C.setopt(C.CURLOPT_POST, 0);
```

11.1.3 perform

```
var string = C.perform()
```

Executes the network operation. The return string contains the answer from the peer.

Notice In case the response is too large the returned string will be truncated. Use the write callback handler in this case to have access to the full reply.

11.1.4 getinfo

```
var string = C.getinfo(C.info)
```

Return informational text about the actions performed, for CURLinfo reference see `http://curl.haxx.se/libcurl/c/curl_easy_getinfo.html`.

Listing 14: Example

```
print(C.getinfo(C.CURLINFO_SIZE_DOWNLOAD) +
    " bytes downloaded in " +
    + C.getinfo(C.CURLINFO_TOTAL_TIME) + " seconds");
```

11.1.5 version

```
var string = C.version()
```

Return the curl library version.

11.1.6 callbacks

The easycurl class provides three callback handlers for operation: the header, write and debug handler.

```
C.header = function(string) {}
C.write = function(string) {}
C.debug = function debug(itype, data) {}
```

11.2 easylist

The easylist class stores option arrays for the libcurl options.

11.2.1 Constructor

```
var CL = new easylist()
```

11.2.2 append

```
CL.append(C.option)
```

11.2.3 toArray

```
var array = CL.toArray()
```

11.3 TcpSocket

The TcpSocket class allows communication to other hosts using TCP protocol. All calls are asynchronous.

Notice The TcpSocket class is deprecated and easycurl should be used instead.

11.3.1 Constructor

```
var T = new TcpSocket()
```

11.3.2 connect

```
T.connect(host, port[, function(success)])
```

Opens a connection to *host*:*port*. When finished the optional function gets called with a boolean argument indicating success.

11.3.3 send

```
T.send(data[, function(bytesSent)])
```

Sends *data* over the previously opened socket. When finished the optional function gets called with the number of bytes actually sent (-1 on error).

11.3.4 receive

```
T.receive(numberOfBytes[,function(data)])
```

Receives *data* over the previously opened socket. When finished the optional function gets called with the data received ("" on error).

11.3.5 receiveLine

```
T.receiveLine(numberOfBytes[, function(data), delimiter])
```

Receives *data* until the sequence Carriage-Return-New-Line is received, or the optional delimiter character is received. When finished the optional function gets called with the data received ("" on error).

11.3.6 close

```
T.close()
```

Closes the socket.

11.3.7 bind

```
var Tin = TcpSocket.bind(port[, function(success)])
```

Opens a server-socket on the given *port*. When finished the optional function gets called with a boolean argument indicating success.

11.3.8 accept

```
Tin.accept(function(socket))
```

Waits for a connection on a previously bound port. The function provided will be called if somebody connects with a new TcpSocket object as argument.

11.3.9 sendTo

```
TcpSocket.sendTo(host, port, data[, function(success)])@
```

Opens a connection to *host*:*port* and sends data to it. When finished the optional function gets called with a boolean argument indicating success.

Static function.