digitalSTROM Server Scripting

digitalSTROM

Version: v1.4-branch[*]

May 22, 2017

[*]Revision: 6d0a8b6a9a98d5874ebfa542a78ea62bf3c04f8d

digitalSTROM Alliance
Building Technology Park Zurich
Brandstrasse 33
CH-8952 Schlieren
Switzerland

Contents

Introduction

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>
```

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
```

Event Documentation

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

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.

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.

Global Functions

Print

Listing 3: print

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

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

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.

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.

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.

Event

Constructor

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

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

raise

```
E.raise()
```

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

Example

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

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.

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.

raise

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

Raises the event and returns the ID.

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.

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.

raise

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

Raises the event and returns the ID.

Example

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

Apartment

Apartment Static Functions

Apartment and data model functions in the global name space.

getName

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

Returns the name of the apartment.

setName

```
setName(name)
```

Sets the name of the apartment.

getDevices

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

Returns a Set that contains all devices of the apartment.

getZones

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

Returns an array that contains all zones of the apartment.

getZoneByID

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

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

getClusters

```
var array = getClusters();
```

Returns an array that contains all used clusters of the apartment.

getDSMeters

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

Returns an array that contains all dSMeters of the apartment.

getDSMeterByDSID

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

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

getConsumption

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

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

getEnergyMeterValue

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

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

getState

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

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

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.

getWeatherInformation

```
var weather = getWeatherInformation();
```

Returns the currently available weather infomration for the location of the dSS. The return object contains the following properties:

TemperatureValue	Current outside temperature value
TemperatureValueTime	Time of last TemperatureValue update
BrightnessValue	Current outside brightness value
BrightnessValueTime	Time of last BrightnessValue update
HumidityValue	Current outside humidity value
HumidityValueTime	Time of last HumidityValue update
WeatherIconId	Weather forecast icon
WeatherConditionId	Weather condition ID
WeatherServiceId	Weather service ID
WeatherServiceTime	Time of last WeatherService infomration update

setWeatherInformation

```
setWeatherInformation(WeatherIconId, WeatherConditionId, WeatherServiceId)
    ;
```

setDeviceVisibility

```
setDeviceVisibility(dsuid, isVisible);
```

Sets the visibility of a TNY device, use a boolean flag for the isVisible parameter.

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
```

Cluster

Cluster objects can be generated by the call Cluster.createCluster(). Use the call Cluster.clusterAddDevice() to add devices to a cluster. Locking and unlocking is implemented in Cluster.clusterConfigurationLock().

Listing 7: Example

```
var colorGroup = 2;
var clusterId = Cluster.createCluster(colorGroup, "clusterName");

Cluster.clusterAddDevice(clusterId, device-dSUID);

var lock = true;
Cluster.clusterConfigurationLock(clusterId, lock);
```

Cluster Properties

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

Property	Type	Description
className	string	=== "Cluster"
id	int	the unique ID of this cluster
name	string	the user name for this device
standardGroup	int	the assigned color group of the cluster
location	int	the cluster location
protectionClass	int	the cluster protection class
floor	int	the cluster floor
automatic	bool	the cluster creation mode. Manual or automatic creation
configurationLock	bool	the cluster configuration lock state
operationLock	bool	the operation lock state of the cluster

Cluster Methods

A cluster object supports the following methods.

createCluster

```
var colorGroup = 2;
var clusterName = "myCluster";
var clusterId = Cluster.createCluster(colorGroup, clusterName);
```

Returns the groupId of the cluster or -1 if cluster could no be generated.

removeCluster

```
var clusterId = 2;
var boolVal = Cluster.removeCluster(clusterId);
```

Returns true, if cluster could be removed, false otherwise.

clusterAddDevice

```
var clusterId = 2;
var deviceId = "3504175fe0000010000004d9";
Cluster.clusterAddDevice(clusterId, deviceId);
```

clusterRemoveDevice

```
var clusterId = 2;
var deviceId = "3504175fe0000010000004d9";
var num = Cluster.clusterRemoveDevice(clusterId, deviceId);
```

clusterConfigurationLock

```
var clusterId = 2;
Cluster.clusterConfigurationLock(clusterId, true);
```

clusterSetName

```
var clusterId = 2;
var name = 'newName';
Cluster.clusterSetName(clusterId, name);
```

clusterGetName

```
var clusterId = 2;
var name = Cluster.clusterGetName(clusterId);
```

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 8: Example

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

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

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.

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

saveScene

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

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

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

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

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

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

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

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

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

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

getConfig

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

getConfigWord

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

setConfig

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

getOutputValue

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

setOutputValue

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

getSensorValue

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

getSensorType

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

getPropertyNode

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

Returns the property tree object of this device.

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 9: Example

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

Set Properties

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

Property	Type	Description
className	string	=== "Set"

Set Methods

A set object supports the following methods.

perform

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

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

Listing 10: Example

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

length

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

combine

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

remove

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

byName

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

byDSID

```
var device = S.byDSID(device—dSID)
```

also accepts a dSUID:

```
var device = S.byDSID(device—dSUID)
```

byShortAddress

```
var device = S.byShortAddress(dSM—dSID, deviceShortAddress)
```

also accepts a dSUID:

```
var device = S.byShortAddress(dSM—dSUID, deviceShortAddress)
```

byFunctionID

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

byZone

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

byDSMeter

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

byGroup

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

byPresence

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

byTag

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

Zone

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

Listing 11: Example

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

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

Zone Methods

A Zone object supports the following methods.

getDevices

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

Returns the list of device objects in this zone.

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

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

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

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.

pushSensorValue

Push a 12-bit raw sensor value to the given group.

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

pushSensorValueFloat

Push a sensor value to the given group. Thows an exception when the given value exceeds the possible value range for the given sensorType.

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

getPropertyNode

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

Returns the property tree object of this zone.

addConnectedDevice

```
Z.addConnectedDevice(groupID)
```

By default, user interfaces only display a group in a zone if there are actual devices connected to it. If a script (e.g. through event subscriptions) simulates a device this call registers the presence of a device in a certain group with the dSS. This call should be repeated for each instance of an attached device.

This registration is not persistent. It has to be done on every startup of the dSS and the script (e.g. on the running event).

removeConnectedDevice

```
Z.removeConnectedDevice(groupID)
```

By default, user interfaces only display a group in a zone if there are actual devices connected to it. If a script (e.g. through event subscriptions) simulates a device this call de-registers the presence of a device in a certain group with the dSS. A script may only use this call if it has called addConnectedDevice() before.

getTemperatureControlStatus

Get the current status of the zone temperature control.

```
var status = Z.getTemperatureControlStatus()
```

Returns an object with the following elements:

ControlMode	Control mode: 0=off; 1=pid-control; 2=zone-follower; 3=fixed-value; 4=manual
OperationMode	Current operation mode of the controller (ControlMode 1, 3, 4)
TemperatureValue	Current temperature of the zone (ControlMode 1)
TemperatureValueTime	Timestamp of last temperature data update (ControlMode 1)
NominalValue	Target temperature of this zone (ControlMode 1)
NominalValueTime	Timestamp of last set point change (ControlMode 1)
ControlValue	Current control value (ControlMode 1, 2, 3, 4)
ControlValueTime	Timestamp of last control value data update (ControlMode 1, 2)

getTemperatureControlConfiguration

Get the configuration of the zone temperature control.

```
var config = Z.getTemperatureControlConfiguration()
```

Returns an object with the following elements:

ControlDSUID	dSUID of the meter or service that runs the control algorithm for this zone, can be zero
ControlMode	Control mode: 0=off; 1=pid-control; 2=zone-follower; 3=fixed-value; 4=manual
ReferenceZone	Zone number of the reference zone (mode 2 only), can be zero
CtrlOffset	Control value offset (mode 2 only)
ManualValue	Fixed control value for manual mode (mode 4 only)
EmergencyValue	Fixed control value in case of malfunction
CtrlKp	Control proportional factor (mode 1 only)
CtrlTs	Control sampling time (mode 1 only)
CtrlTi	Control integrator time constant (mode 1 only)
CtrlKd	Control differential factor (mode 1 only)
CtrlImin	Control minimum integrator value (mode 1 only)
CtrlImax	Control maximum integrator value (mode 1 only)
CtrlYmin	Control minimum control value (mode 1 only)
CtrlYmax	Control maximum control value (mode 1 only)
CtrlAntiWindUp	Control integrator anti wind up: 0=inactive, 1=active (mode 1 only)
CtrlKeepFloorWarm	Control mode with higher priority on comfort: 0=inactive, 1=active (mode 1 only)

getTemperatureControlValues

Get the temperature control operation mode preset values for a zone. Every control operation mode has up to 15 presets defined.

```
var values = Z.getTemperatureControlValues()
```

Returns an object with the following elements:

Off	Preset value for operation mode 0: "Off"
Comfort	Preset value for operation mode 1: "Comfort"
Economy	Preset value for operation mode 2: "Economy"
NotUsed	Preset value for operation mode 3: "Not Used"
Night	Preset value for operation mode 4: "Night"
Holiday	Preset value for operation mode 5: "Holiday"

setTemperatureControlConfiguration

Get the temperature control operation mode preset values for a zone. Every control operation mode has up to 15 presets defined.

```
Z.setTemperatureControlConfiguration(ControlDSUID, configObject)
```

The configObject can have the following parameters. Missing parameter will not be changed.

ControlMode	Control mode, can be one of: 0=off; 1=pid-control; 2=zone-follower; 3=fixed-value; 4=manual
ReferenceZone	Zone number of the reference zone
CtrlOffset	Control value offset
EmergencyValue	Fixed control value in case of malfunction
ManualValue	Control value for manual mode
CtrlKp	Control proportional factor
CtrlTs	Control sampling time
CtrlTi	Control integrator time constant
CtrlKd	Control differential factor
CtrlImin	Control minimum integrator value
CtrlImax	Control maximum integrator value
CtrlYmin	Control minimum control value
CtrlYmax	Control maximum control value
CtrlAntiWindUp	Control integrator anti wind up
CtrlKeepFloorWarm	Control mode with higher priority on comfort

setTemperatureControlValues

Set the temperature control operation mode preset values for a zone. Single values can be given and the others that do not change may be omitted.

```
Z.setTemperatureControlValues(configObject)
```

The configObject can have the following parameters. Missing parameter will not be changed.

Off	Preset value for operation mode 0: "Off"
Comfort	Preset value for operation mode 1: "Comfort"
Economy	Preset value for operation mode 2: "Economy"
NotUsed	Preset value for operation mode 3: "Not Used"
Night	Preset value for operation mode 4: "Night"
Holiday	Preset value for operation mode 5: "Holiday"

getAssignedSensor

Set the temperature control operation mode preset values for a zone. Single values can be given and the others that do not change may be omitted.

```
var sensor = Z.getAssignedSensor(sensorType)
```

Returns an object with the following elements:

sensorType	Numerical value of the sensor type
dsuid	dSUID of the source device

State

State objects represent an apartment state. The state object can be generated by the global function getState().

```
var S = getState()
```

State Properties

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

Property	Type	Description
className	string	=== "State"
name	string	the name for this state
type	string	
value	string	

State Methods

A State object supports the following methods.

getValue

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

setValue

```
S.setValue(value, origin)
```

getPropertyNode

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

Returns the property tree object of this state.

dSMeter

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

Listing 12: Example

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

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

dSMeter Methods

A dSMeter object supports the following methods.

getPowerConsumption

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

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

getEnergyMeterValue

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

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

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.

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.

getPropertyNode

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

Returns the property tree object of this dSMeter.

Metering

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

Metering Static Methods

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".

getSeries

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

Returns the stored series.

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

Property

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

Property Static Functions

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.

getProperty

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

Returns the value at *propPath*.

Listing 13: Example

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

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.

removeListener

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

Removes a previously installed listener.

hasFlag

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

Returns true if the given flag is set.

setFlag

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

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

getNode

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

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

store

```
Property.store()
```

Persists the properties of the private subtree of the script.

load

```
Property.load()
```

Loads previously persisted properties.

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.

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.

setValue

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

Sets the value of the property to *value*.

setStatusValue

```
P.setStatusValue(value)
```

Sets the value of a status node *value*. Checks that the status property actually belongs to this script and that the *value* is within the defined value range.

setListener

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

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

removeListener

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

Removes a previously registered callback.

getChild

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

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

getChildren

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

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

getParent

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

Returns the parent property object.

removeChild

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

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

hasFlag

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

Returns true if the given flag is set.

setFlag

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

Sets the boolean *value* of the flag.

getName

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

Returns the name of the property object.

getPath

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

Returns the full path of the property object.

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.

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/.

Constructor

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

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 14: Example

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

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.

perform_async

```
var string = C.perform_async(function(success) { ... })
```

Executes the network operation asynchronously. The callback function is called when the request has completed with a boolean parameter *success*.

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 15: Example

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

getdate

```
var timestamp = C.getdate(dateString)
```

Returns number of seconds since 1.1.1970 for the date given in parameter *dateString*. Plain wrapper around curl_getdate (http://curl.haxx.se/libcurl/c/curl_getdate.html).

version

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

Return the curl library version.

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) {}
```

easylist

The easylist class stores option arrays for the libcurl options.

Constructor

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

append

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

toArray

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

Webservice Connection

For add-ons that require to communicate with the digitalSTROM webservice, a special interface is provided. This interface is only active if remote connectivity has been enabled by the user (i.e. the user has created a my.digitalSTROM account and has activated the connection to his dSS).

Constructor

```
var handle = new WebserviceConnection();
```

Simple Request

```
handle.simplerequest(serviceCallURL, URLparams, requestType, authenticated
    [, function(code,data)])
```

Send a "simple" request to the webservice, usually something where specifying an URL is sufficient (i.e. no extra headers or postdata is needed).

Parameters:

- serviceCallURL: relative URL of the webservice call
- URLParams: a string of key value pairs representing the URL parameters, i.e. key1=value1&key2=valu2
- requestType: HTTP request type, either "GET" or "POST"
- authenticated: boolean flag indicating if default authentication parameters (i.e. token=osptoken&dssid=dsid) should be automatically appended to the parameters
- function(code, data): optional callback function that will be executed once the asynchronous call goes through, "code" will contain the HTTP response code, "data" will contain the raw data as returned by the webservice.

Request

```
handle.request(serviceCallURL,URLparams, requestType, headerObj, formpost,
        authenticated [,function(code,data)])
```

Send a request to the webservice, allows to specify extra HTTP headers as well as formpost data.

Parameters:

- serviceCallURL: relative URL of the webservice call
- URLParams: a string of key value pairs representing the URL parameters, i.e. key1=value1&key2=valu2
- requestType: HTTP request type, either "GET" or "POST"
- headerObj: javascript object representing the HTTP headers, i.e.: { "HEADER1" : "header1value", "HEADER2" : "header2value" }
- formpost: string representing the raw formpost data
- authenticated: boolean flag indicating if default authentication parameters (i.e. token=osptoken&dssid=dsid) should be automatically appended to the parameters
- function(code, data): optional callback function that will be executed once the asynchronous call goes through, "code" will contain the HTTP response code, "data" will contain the raw data as returned by the webservice.

SQL Database

This class allows SQL database access.

Notice SQL class for scripting is available since dSS release 1.32.0.

Constructor

```
var db = new SQL();
```

query

```
db.query("select * from mytable;");
```

Each script is assigned an own database, this function allows to access the database via SQL.