**JETI DC/DS Lua Programming API**

**Introduction**

The DC/DS-24 transmitter line brings new possibilities and enhancements of individual user programming due to Lua extension language. From now on, the capabilities of the transmitter become almost unlimited and it is only up to the user’s imagination what can be done with the transmitter.

Lua is a powerful, fast, lightweight, embeddable scripting language. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

The transmitter is able to handle up to **10 Lua applications simultaneously**. Each application can offer up to **two telemetry entries** (either presented by a small window or overwriting the whole Desktop). To provide easy interaction and configuration, every application can offer up to **two forms** within the transmitter menu structures.

The DC/DS-16 and DC/DS-14 transmitters are able to run up to **2 applications** simultaneously and can offer up to **4 Lua controls**. These transmitters have a strict memory limitation which is set to 50kB. If the application tries to allocate more memory, the whole Lua environment will be disabled.

More about Lua (Currently in use: Lua 5.3.1):

- [http://www.lua.org/](http://www.lua.org/)
- [http://www.lua.org/manual/5.3/](http://www.lua.org/manual/5.3/)

Lua is compiled with the following parameters:

- **LUA_32BITS** – tells the Lua interpreter that the size of integer and floating point numbers is always 32 bits. The interpreter can benefit from hardware FPU support.
- **LUA_FLOORN2I** – floating point number is always floored to the nearest integer if the called function requires an integer data type.
- Compatibility with older Lua versions (5.2 and 5.1) has been excluded from the build.

**WARNING**

Do not use Lua applications for controlling any model function that could cause a crash if the application misbehaves or stops executing.

**Directories**

The Lua applications must be placed on the internal SD card, into the /Apps folder. The applications inside this folder are automatically loaded during transmitter startup. The filenames must comply with the 8.3 format and must have a .lua extension (e.g. SCRIPT1.LUA). Based on the application filename, a unique 32-bit identifier is created, so that the system can reference each application quite easily. If the application filename changes, all model configuration for that application will be lost (e.g. telemetry screens, model and system setup).
You can use the ability to load the external Lua libraries using `require "modname"` statement. The libraries must be placed in a directory according to one of these schemes:

- `/Apps/lib/<modname>.lua`
- `/Apps/lib/<modname>/init.lua`
- `/Apps/<modname>.lua`
- `/Apps/<modname>/init.lua`

### Libraries

<table>
<thead>
<tr>
<th>Lua Standard Libraries</th>
<th>Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>package</td>
<td>YES (loadlib not supported)</td>
</tr>
<tr>
<td>coroutine</td>
<td>NO</td>
</tr>
<tr>
<td>table</td>
<td>YES</td>
</tr>
<tr>
<td>io</td>
<td>YES (limited)</td>
</tr>
<tr>
<td>os</td>
<td>NO</td>
</tr>
<tr>
<td>string</td>
<td>YES</td>
</tr>
<tr>
<td>legacy bit32</td>
<td>NO (bitwise operators are standard in Lua 5.3)</td>
</tr>
<tr>
<td>math</td>
<td>YES</td>
</tr>
<tr>
<td>debug</td>
<td>NO</td>
</tr>
</tbody>
</table>

### Additional custom libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Basic system features, audio playback.</td>
</tr>
<tr>
<td>lcd</td>
<td>Drawing primitives, texts and images on screen.</td>
</tr>
<tr>
<td>form</td>
<td>User interaction using standard form dialogue.</td>
</tr>
<tr>
<td>dir</td>
<td>Allows simple traversing through directories.</td>
</tr>
<tr>
<td>json</td>
<td>Functions for JSON encoding and decoding.</td>
</tr>
</tbody>
</table>

### Hardware specification

The transmitter runs at this configuration (DC/DS-24):

- MCU: STM32F439 @ 168MHz
- External memory: 8MB (1MB reserved for framebuffer and system resources)
- SD card support: Up to 32GB micro SDHC
- Audio playback: MP3 (44.1kHz, 32kHz), WAV (8kHz, 11kHz, 16kHz, 22.05kHz, 32kHz, 44.1kHz; Mono/Stereo)
- Vibration support: Left and right gimbal

DC/DS-16:

- MCU: STM32F405 @ 168MHz
- SD card support: Up to 32GB micro SDHC
- Audio playback: WAV (8kHz, 11kHz, 16kHz, 22.05kHz, 32kHz, 44.1kHz; Mono/Stereo)
Application interface

Every application consists of a single Lua script located in the /Apps folder. Additional scripts and Lua libraries can be loaded using the require statement. The application script must return an array which describes its interface.

---

-- App1.lua
-- Application initialization.
local function init(code)
    print (“Application initialized”)
end

-- Loop function is called in regular intervals
local function loop()
end

-- Application interface
return {init = init, loop = loop, author = “JETI model”, version = “1.0”,
        name = “App name”}

---

In this example we defined a simple application that prints a text to the debug window (available through the menu Applications – User Applications):

- **init** (<code>) – function that is called every time the model is loaded or changed. Here you can initialize all variables as well as register telemetry windows and configuration forms. (Limitation: access to the lcd and form libraries is disabled)
- **loop** – function that is called in regular intervals, every 20-30ms. The loop time is not guaranteed. (Limitation: access to the lcd is disabled, access to the form library limited)
- **author** – name of the application author
- **version** – string description of the application version
- **name** – application name

The **init** (<code>) function is called with a single parameter, “code”, which specifies a moment the application is loaded/initialized:

- code = 0 – the application has been loaded without any previous information.
- code = 1 – the application was loaded after the model had been loaded or changed.
- code = 2 – the application was loaded after the transmitter had disconnected from USB.

The <code> parameter has been available since V4.20.
Global/local variables

All Lua applications share the same system environment. If you define any global variable or function, it will be accessible from all other Lua applications.

-- The counter variable is defined locally for a given file.
local counter = 1

-- Loop function is called in regular intervals. Defined locally for a given file.
local function loop()
    -- Local variable temp
    local temp = counter + 1
    print (temp)
    counter = temp
end

RECOMMENDATION

Always use local variables and functions. Using this approach, you will prevent possible problems if the system runs several Lua applications simultaneously. Even variables defined inside local functions without local keyword are global.
Loading the applications

The DC/DS-24 offers a simple overview of installed and active applications. See the menu Applications – User Applications. Every time you switch to a different model, the Lua context resets and the selected applications are reloaded.

The application overview shows the application name, version, maximum CPU utilization [%] and status. If you press the 3D button over the application name, you will be redirected to the registered application form (if available).

The F(1) “CMD” button redirects you to the debug console.

Using the F(3) “Plus” button you can activate additional Lua applications (up to 10 can be active per model at the same time).

After pressing the F(4) “Delete” button the selected application will be uninstalled from the model. This operation only clears the application entries from the model memory and will not alter any file located on the SD card.

Debug console

The F(1) “Info” button allows you to see the Lua garbage collector’s total size (visible only via the PC console).

After pressing the F(3) “Refresh” button, all the installed Lua applications will be reloaded. The Lua application context will be destroyed and recreated.

The F(4) “Clr” button clears texts of the debug console.
Global variables/system constants
This section specifies the global variables available to the user.

<table>
<thead>
<tr>
<th>General values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLED</td>
</tr>
<tr>
<td>ENABLED</td>
</tr>
<tr>
<td>HIGHLIGHTED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Font definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FONT_NORMAL</td>
</tr>
<tr>
<td>FONT_BOLD</td>
</tr>
<tr>
<td>FONT_MINI</td>
</tr>
<tr>
<td>FONT_BIG</td>
</tr>
<tr>
<td>FONT_MAXI</td>
</tr>
<tr>
<td>FONT_REVERSED</td>
</tr>
<tr>
<td>FONT_GRAYED</td>
</tr>
<tr>
<td>FONT_XOR</td>
</tr>
<tr>
<td>FONT_OR</td>
</tr>
<tr>
<td>FONT_AND</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Menu definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MENU_NONE</td>
</tr>
<tr>
<td>MENU_MAIN</td>
</tr>
<tr>
<td>MENU_FINE</td>
</tr>
<tr>
<td>MENU_ADVANCED</td>
</tr>
<tr>
<td>MENU_APPS</td>
</tr>
<tr>
<td>MENU_SYSTEM</td>
</tr>
<tr>
<td>MENU_GAMES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Persistent data specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
</tr>
<tr>
<td>MODEL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Audio definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIO_BACKGROUND</td>
</tr>
<tr>
<td>AUDIO_IMMEDIATE</td>
</tr>
<tr>
<td>AUDIO_QUEUE</td>
</tr>
<tr>
<td>SOUND_START</td>
</tr>
<tr>
<td>SOUND_BOUND</td>
</tr>
<tr>
<td>SOUND_LOWTXVOLT</td>
</tr>
<tr>
<td>SOUND_LOWSIGNAL</td>
</tr>
<tr>
<td>SOUND_SIGNALLOSS</td>
</tr>
<tr>
<td>SOUND_RANGETEST</td>
</tr>
<tr>
<td>SOUND_AUTOTRIM</td>
</tr>
<tr>
<td>SOUND_INACT</td>
</tr>
<tr>
<td>SOUND_LOWQ</td>
</tr>
<tr>
<td>SOUND_RXRESET</td>
</tr>
</tbody>
</table>
### Key definitions

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_1</td>
<td>F(1) button code.</td>
</tr>
<tr>
<td>KEY_2</td>
<td>F(2) button code.</td>
</tr>
<tr>
<td>KEY_3</td>
<td>F(3) button code.</td>
</tr>
<tr>
<td>KEY_4</td>
<td>F(4) button code.</td>
</tr>
<tr>
<td>KEY_5</td>
<td>F(5) button code.</td>
</tr>
<tr>
<td>KEY_MENU</td>
<td>Menu button code.</td>
</tr>
<tr>
<td>KEY_ESC</td>
<td>Escape button code.</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter (Rotary pressed) button code.</td>
</tr>
<tr>
<td>KEY_UP</td>
<td>Rotary Up button code.</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>Rotary Down button code.</td>
</tr>
<tr>
<td>KEY_RELEASED</td>
<td>Keycode generated after any button has been released.</td>
</tr>
</tbody>
</table>
### Default system images

<table>
<thead>
<tr>
<th>Image Code</th>
<th>Image</th>
<th>Image Code</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;:sndOn&quot;</td>
<td>⏯️</td>
<td>&quot;:inc&quot;</td>
<td>+</td>
</tr>
<tr>
<td>&quot;:sndOff&quot;</td>
<td>⏰</td>
<td>&quot;:dec&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;:left&quot;</td>
<td>←</td>
<td>&quot;:music&quot;</td>
<td>🎸</td>
</tr>
<tr>
<td>&quot;:right&quot;</td>
<td>➔</td>
<td>&quot;:folder&quot;</td>
<td>🗄️</td>
</tr>
<tr>
<td>&quot;:up&quot;</td>
<td>↑</td>
<td>&quot;:global&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;:down&quot;</td>
<td>↓</td>
<td>&quot;:single&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;:ok&quot;</td>
<td>✔️</td>
<td>&quot;:rec&quot;</td>
<td>●</td>
</tr>
<tr>
<td>&quot;:edit&quot;</td>
<td>⌨️</td>
<td>&quot;:stopSmall&quot;</td>
<td>■</td>
</tr>
<tr>
<td>&quot;:okBig&quot;</td>
<td>✔️</td>
<td>&quot;:refresh&quot;</td>
<td>🔌</td>
</tr>
<tr>
<td>&quot;:listBig&quot;</td>
<td>▼</td>
<td>&quot;:tools&quot;</td>
<td>✘</td>
</tr>
<tr>
<td>&quot;:cross&quot;</td>
<td>✗</td>
<td>&quot;:add&quot;</td>
<td>+</td>
</tr>
<tr>
<td>&quot;:wait&quot;</td>
<td>✗</td>
<td>&quot;:delete&quot;</td>
<td>✗</td>
</tr>
<tr>
<td>&quot;:crossBig&quot;</td>
<td>✗</td>
<td>&quot;:graphBig&quot;</td>
<td>⬆️</td>
</tr>
<tr>
<td>&quot;:list&quot;</td>
<td>▼</td>
<td>&quot;:graph&quot;</td>
<td>⬇️</td>
</tr>
<tr>
<td>&quot;:modelG&quot;</td>
<td>🎶</td>
<td>&quot;:forward&quot;</td>
<td>→→</td>
</tr>
<tr>
<td>&quot;:modelGDS&quot;</td>
<td>🎮</td>
<td>&quot;:backward&quot;</td>
<td>←←</td>
</tr>
<tr>
<td>&quot;:modelAir&quot;</td>
<td>♂</td>
<td>&quot;:zoomIn&quot;</td>
<td>📀</td>
</tr>
<tr>
<td>&quot;:modelHeli&quot;</td>
<td>♂</td>
<td>&quot;:zoomOut&quot;</td>
<td>📀</td>
</tr>
<tr>
<td>&quot;:modelCopter&quot;</td>
<td>♂</td>
<td>&quot;:key&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;:servo&quot;</td>
<td>🎫</td>
<td>&quot;:file&quot;</td>
<td>📄</td>
</tr>
<tr>
<td>&quot;:play&quot;</td>
<td>⏯️</td>
<td>&quot;:timer&quot;</td>
<td>⌚</td>
</tr>
<tr>
<td>&quot;:stop&quot;</td>
<td>⏰</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Supported charset

The DC/DS-24 supports a subset of UTF-8 charset. A standard 7-bit ASCII is supported, plus additional Unicode characters which are listed in the table below. Since Lua 5.3 contains native UTF-8 support for strings, you can write the applications directly in UTF-8 encoding (this is a preferred option as well).

```lua
-- This function displays some text with special (Unicode) characters
local function printForm()
    local altitude = 120
    local temperature = 25
    lcd.drawText(10,10,"»Altitude« " .. altitude .. "m")
    lcd.drawText(10,30,"»Temperature« " .. temperature .. "°C")
end
-- During initialization the application registers a form inside the main menu
local function init()
    system.registerForm(1,MENU_MAIN,"Test 1 - Special chars",nil, nil,printForm)
end

return {init=init,author="JETI model", version="1.0"}
```

![Image showing the display output of the printForm function]
Available special characters

<table>
<thead>
<tr>
<th>Unicode</th>
<th>Character</th>
<th>Unicode</th>
<th>Character</th>
<th>Unicode</th>
<th>Character</th>
<th>Unicode</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>171</td>
<td>«</td>
<td>218</td>
<td>Ú</td>
<td>248</td>
<td>ø</td>
<td>327</td>
<td>Ń</td>
</tr>
<tr>
<td>176</td>
<td>°</td>
<td>219</td>
<td>Ú</td>
<td>249</td>
<td>ù</td>
<td>328</td>
<td>ņ</td>
</tr>
<tr>
<td>177</td>
<td>±</td>
<td>220</td>
<td>Ú</td>
<td>250</td>
<td>ú</td>
<td>336</td>
<td>Ō</td>
</tr>
<tr>
<td>187</td>
<td>»</td>
<td>221</td>
<td>Ý</td>
<td>251</td>
<td>û</td>
<td>337</td>
<td>ō</td>
</tr>
<tr>
<td>192</td>
<td>Å</td>
<td>222</td>
<td>þ</td>
<td>252</td>
<td>ü</td>
<td>340</td>
<td>Ř</td>
</tr>
<tr>
<td>193</td>
<td>Á</td>
<td>223</td>
<td>ß</td>
<td>253</td>
<td>ý</td>
<td>341</td>
<td>ř</td>
</tr>
<tr>
<td>194</td>
<td>Ä</td>
<td>224</td>
<td>à</td>
<td>254</td>
<td>þ</td>
<td>344</td>
<td>Ř</td>
</tr>
<tr>
<td>195</td>
<td>Ä</td>
<td>225</td>
<td>á</td>
<td>255</td>
<td>ÿ</td>
<td>345</td>
<td>ř</td>
</tr>
<tr>
<td>196</td>
<td>Ä</td>
<td>226</td>
<td>â</td>
<td>260</td>
<td>Â</td>
<td>346</td>
<td>Š</td>
</tr>
<tr>
<td>197</td>
<td>Á</td>
<td>227</td>
<td>å</td>
<td>261</td>
<td>ã</td>
<td>347</td>
<td>š</td>
</tr>
<tr>
<td>198</td>
<td>Æ</td>
<td>228</td>
<td>ā</td>
<td>262</td>
<td>Ć</td>
<td>350</td>
<td>Š</td>
</tr>
<tr>
<td>199</td>
<td>Ç</td>
<td>229</td>
<td>à</td>
<td>263</td>
<td>Ă</td>
<td>351</td>
<td>š</td>
</tr>
<tr>
<td>200</td>
<td>Ë</td>
<td>230</td>
<td>æ</td>
<td>268</td>
<td>Ń</td>
<td>352</td>
<td>Š</td>
</tr>
<tr>
<td>201</td>
<td>É</td>
<td>231</td>
<td>č</td>
<td>269</td>
<td>Ė</td>
<td>353</td>
<td>š</td>
</tr>
<tr>
<td>202</td>
<td>È</td>
<td>232</td>
<td>ē</td>
<td>270</td>
<td>Ń</td>
<td>354</td>
<td>Ť</td>
</tr>
<tr>
<td>203</td>
<td>Ê</td>
<td>233</td>
<td>ņ</td>
<td>271</td>
<td>d’</td>
<td>355</td>
<td>t</td>
</tr>
<tr>
<td>204</td>
<td>İ</td>
<td>234</td>
<td>Ė</td>
<td>272</td>
<td>D</td>
<td>356</td>
<td>Ŭ</td>
</tr>
<tr>
<td>205</td>
<td>I</td>
<td>235</td>
<td>Ī</td>
<td>273</td>
<td>d</td>
<td>357</td>
<td>t’</td>
</tr>
<tr>
<td>206</td>
<td>İ</td>
<td>236</td>
<td>Į</td>
<td>280</td>
<td>Ê</td>
<td>366</td>
<td>Ū</td>
</tr>
<tr>
<td>207</td>
<td>Ī</td>
<td>237</td>
<td>Ī</td>
<td>281</td>
<td>Ė</td>
<td>367</td>
<td>ŭ</td>
</tr>
<tr>
<td>209</td>
<td>Ñ</td>
<td>238</td>
<td>î</td>
<td>282</td>
<td>Ė</td>
<td>368</td>
<td>Ū</td>
</tr>
<tr>
<td>210</td>
<td>Ô</td>
<td>239</td>
<td>ì</td>
<td>283</td>
<td>Ė</td>
<td>369</td>
<td>ŭ</td>
</tr>
<tr>
<td>211</td>
<td>Ö</td>
<td>240</td>
<td>δ</td>
<td>313</td>
<td>Ė</td>
<td>377</td>
<td>Ž</td>
</tr>
<tr>
<td>212</td>
<td>Ő</td>
<td>241</td>
<td>Ň</td>
<td>317</td>
<td>L</td>
<td>378</td>
<td>Ž</td>
</tr>
<tr>
<td>213</td>
<td>Œ</td>
<td>242</td>
<td>ò</td>
<td>318</td>
<td>l’</td>
<td>379</td>
<td>Ž</td>
</tr>
<tr>
<td>214</td>
<td>Ô</td>
<td>243</td>
<td>ó</td>
<td>321</td>
<td>L</td>
<td>380</td>
<td>Ž</td>
</tr>
<tr>
<td>215</td>
<td>×</td>
<td>244</td>
<td>ô</td>
<td>322</td>
<td>l</td>
<td>381</td>
<td>Ž</td>
</tr>
<tr>
<td>216</td>
<td>Ø</td>
<td>245</td>
<td>Õ</td>
<td>323</td>
<td>Ň</td>
<td>382</td>
<td>Ž</td>
</tr>
<tr>
<td>217</td>
<td>Ů</td>
<td>246</td>
<td>Ũ</td>
<td>324</td>
<td>ň</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# System library

## Functions overview

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCPU</td>
<td>Gets the CPU utilization (0-100%).</td>
</tr>
<tr>
<td>getTime</td>
<td>Gets the current time as seconds elapsed since Jan 1 2000, 00:00:00.</td>
</tr>
<tr>
<td>getTimeCounter</td>
<td>Gets current timestamp in milliseconds.</td>
</tr>
<tr>
<td>getDateDateTime</td>
<td>Retrieves the current date and time in a table.</td>
</tr>
<tr>
<td>getVersion</td>
<td>Gets the transmitter SW version (e. g. “4.00”).</td>
</tr>
<tr>
<td>getDeviceType</td>
<td>Gets the device type as string (“JETI DC-24”).</td>
</tr>
<tr>
<td>getLocale</td>
<td>Gets the current locale (e. g. “en”).</td>
</tr>
<tr>
<td>getTXTelemetry</td>
<td>Retrieves the system voltage, signal quality etc.</td>
</tr>
<tr>
<td>getUserName</td>
<td>Gets the user name.</td>
</tr>
<tr>
<td>getSerialCode</td>
<td>Gets the registration number.</td>
</tr>
<tr>
<td>getSensors</td>
<td>Retrieves all detected sensors/values in a table.</td>
</tr>
<tr>
<td>getSensorByID</td>
<td>Gets a single sensor value.</td>
</tr>
<tr>
<td>getInputs</td>
<td>Gets multiple values of sticks/switches.</td>
</tr>
<tr>
<td>getInputsVal</td>
<td>Gets the current value based on a &quot;SwitchItem&quot; datatype. Up to 8 switches can be defined.</td>
</tr>
<tr>
<td>getRawIMU</td>
<td>Gets raw gyro/accelerometer readings (DS only)</td>
</tr>
<tr>
<td>getIMU</td>
<td>Gets calculated position/acceleration (DS only)</td>
</tr>
<tr>
<td>messageBox</td>
<td>Invokes the info/alert message pop-up.</td>
</tr>
<tr>
<td>registerTelemetry</td>
<td>Registers a new telemetry window.</td>
</tr>
<tr>
<td>registerForm</td>
<td>Registers a new interactive form.</td>
</tr>
<tr>
<td>unregisterTelemetry</td>
<td>Unregisters a given telemetry window.</td>
</tr>
<tr>
<td>unregisterForm</td>
<td>Unregisters a given form.</td>
</tr>
<tr>
<td>registerControl</td>
<td>Registers an output control.</td>
</tr>
<tr>
<td>setControl</td>
<td>Sets a value to the output control.</td>
</tr>
<tr>
<td>unregisterControl</td>
<td>Unregisters a given output control.</td>
</tr>
<tr>
<td>pLoad</td>
<td>Loads and registers a permanent parameter.</td>
</tr>
<tr>
<td>pSave</td>
<td>Saves the permanent parameter.</td>
</tr>
<tr>
<td>vibration</td>
<td>Starts vibrations.</td>
</tr>
<tr>
<td>playFile</td>
<td>Plays a specified audio file.</td>
</tr>
<tr>
<td>playNumber</td>
<td>Announces a numeric value by voice.</td>
</tr>
<tr>
<td>playBeep</td>
<td>System beep.</td>
</tr>
<tr>
<td>playSystemSound</td>
<td>Plays one of the system sounds.</td>
</tr>
<tr>
<td>isPlayback</td>
<td>Checks if any audio file is being played.</td>
</tr>
<tr>
<td>stopPlayback</td>
<td>Stops the playback.</td>
</tr>
<tr>
<td>setProperty</td>
<td>Sets some of the system properties.</td>
</tr>
<tr>
<td>getProperty</td>
<td>Retrieves one of the available system properties. (since V4.20)</td>
</tr>
<tr>
<td>getSwitchInfo</td>
<td>Gets information about the assigned switch (since V4.22)</td>
</tr>
</tbody>
</table>
**LCD library**

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Display width in px (320)</td>
</tr>
<tr>
<td>height</td>
<td>Display height in px (240)</td>
</tr>
</tbody>
</table>

**Functions overview**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setColor</td>
<td>Sets the foreground color to a new value.</td>
</tr>
<tr>
<td>drawPoint</td>
<td>Draws a point with given coordinates.</td>
</tr>
<tr>
<td>drawLine</td>
<td>Draws a line.</td>
</tr>
<tr>
<td>drawText</td>
<td>Draws a text with specified font.</td>
</tr>
<tr>
<td>drawRectangle</td>
<td>Draws a rectangle (optionally rounded).</td>
</tr>
<tr>
<td>drawFilledRectangle</td>
<td>Draws filled rectangle.</td>
</tr>
<tr>
<td>drawCircle</td>
<td>Draws a circle. <em>(since V4.20)</em></td>
</tr>
<tr>
<td>drawEllipse</td>
<td>Draws an ellipse. <em>(since V4.20)</em></td>
</tr>
<tr>
<td>drawNumber</td>
<td>Draws a number (integer only).</td>
</tr>
<tr>
<td>drawImage</td>
<td>Draws an image loaded by loadImage().</td>
</tr>
<tr>
<td>loadImage</td>
<td>Loads an image from SD card (JPG or PNG).</td>
</tr>
<tr>
<td>getTextHeigt</td>
<td>Gets the text height.</td>
</tr>
<tr>
<td>getTextWidth</td>
<td>Gets the text width.</td>
</tr>
<tr>
<td>getBgColor</td>
<td>Gets the background color.</td>
</tr>
<tr>
<td>getFgColor</td>
<td>Gets the foreground color.</td>
</tr>
<tr>
<td>setClipping</td>
<td>Sets a clipping rectangle. <em>(since V4.20)</em></td>
</tr>
<tr>
<td>resetClipping</td>
<td>Resets already defined clipping rectangle. <em>(since V4.20)</em></td>
</tr>
</tbody>
</table>
Form library
The form library is available only when the interactive Lua form is opened.

Functions overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Description________________________________________________________________________________________________________________________________________________________________________________________________________</th>
</tr>
</thead>
<tbody>
<tr>
<td>addRow</td>
<td>Creates a new row and pushes it to the layout.</td>
</tr>
<tr>
<td>addSpacer</td>
<td>Creates a blank space with specified dimensions.</td>
</tr>
<tr>
<td>addLabel</td>
<td>Creates a new label and pushes it to the layout.</td>
</tr>
<tr>
<td>addIntbox</td>
<td>Creates a new integer editor and pushes it to the layout.</td>
</tr>
<tr>
<td>addTextbox</td>
<td>Creates a new text editor and pushes it to the layout.</td>
</tr>
<tr>
<td>addSelectbox</td>
<td>Creates a drop-down menu and pushes it to the layout.</td>
</tr>
<tr>
<td>addAudioFilebox</td>
<td>Creates a drop-down menu of available audio files and pushes it to the layout.</td>
</tr>
<tr>
<td>addInputbox</td>
<td>Creates an input selection control and pushes it to the layout.</td>
</tr>
<tr>
<td>addCheckbox</td>
<td>Creates a new checkbox element.</td>
</tr>
<tr>
<td>addLink</td>
<td>Creates a link control and pushes it to the layout.</td>
</tr>
<tr>
<td>getValue</td>
<td>Gets the value of a specified control.</td>
</tr>
<tr>
<td>setValue</td>
<td>Sets the value of a specified control.</td>
</tr>
<tr>
<td>setProperties</td>
<td>Sets the properties of a specified control.</td>
</tr>
<tr>
<td>setButton</td>
<td>Sets the button type and label for F(1) – F(5)</td>
</tr>
<tr>
<td>getButton</td>
<td>Gets the button type and label (F(1) – F(5))</td>
</tr>
<tr>
<td>getActiveForm</td>
<td>Gets the ID of an active form (or nil)</td>
</tr>
<tr>
<td>close</td>
<td>Closes the form.</td>
</tr>
<tr>
<td>reinit</td>
<td>Destroys all created components and reinitializes the form.</td>
</tr>
<tr>
<td>preventDefault</td>
<td>Prevents default system behavior after a key has been pressed.</td>
</tr>
<tr>
<td>waitForRelease</td>
<td>Forces all the buttons to be released before any other button can be pressed.</td>
</tr>
<tr>
<td>getFocusedRow</td>
<td>Returns the currently focused row.</td>
</tr>
<tr>
<td>setFocusedRow</td>
<td>Sets the focused row in the form.</td>
</tr>
<tr>
<td>addIcon</td>
<td>Adds an image into a form. <em>(since V4.20)</em></td>
</tr>
<tr>
<td>setTitle</td>
<td>Sets the form title. <em>(since V4.20)</em></td>
</tr>
<tr>
<td>question</td>
<td>Raises the YES/NO question <em>(since V4.20)</em></td>
</tr>
</tbody>
</table>
Library functions reference

dir (<path>)
The dir function allows you to traverse through a directory within a single cycle. After calling this function for the first time, a directory iterator is created. Any succeeding call returns the next directory entry.

Parameters
- path (string) – a specified path located on the SD card. Relative or absolute paths can be used. The path should be accessible.

Return value
- <itemName> (string), <ItemType> (string), <fileSize> (integer) - entry details. Item type can be either “file” or “folder”. File size is specified in Bytes.
- nil – in case of error or directory end.

Examples

-- Print all files and directories inside the Lang folder
local function init()
    for name, filetype, size in dir("/Lang") do
        print(name, filetype, size)
    end
end
-------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.0"}
**io.open (<path> [, <mode = "r">])**

The io library has been simplified and currently supports a subset of the full io library.

The open () function opens a specified file located on the SD card. After calling the io.open () function you can call subsequent reading and writing functions. As soon as you finish the file operations, the io.close () function must be called to clear the resources.

**RECOMMENDATION**
If you have several model-specific parameters that need to be synchronized, please use the `system.pLoad()` and `system.pSave()` functions instead io library functions.

**Parameters**
- **path** (string) – a specified file path located on the SD card. Only absolute paths are used. The path should be accessible.
- **mode** (string) – a file access option. Can be one of the following strings:
  - "r" – read access. The open () function opens an existing file for reading. The read pointer is located at the beginning of the file.
  - "w" – write access. The open () function opens existing file or creates a new one if it didn't exist. The file is truncated to zero length.
  - "a" – append/write access. The open () function opens existing file or creates a new one if it didn't exist. The write pointer is located at the end of the file keeping its existing contents.

**Return value**
- `<fileObject>` - an opened file descriptor in case of success.
- `nil` – in case of error.

**Examples**
See `io.read (<fileObject>, <noBytes>)`
**io.close (<fileObject>)**

The close () function closes a file previously opened by the io.open () function.

**Parameters**
- **fileObject** – a file descriptor returned by the io.open () function.

**Return value**
none

**Examples**
See `io.read (<fileObject>, <noBytes>)`
**io.read (<fileObject>, <noBytes>)**

The read () function reads specified number of bytes from an opened file.

**Parameters**

- **fileObject** – a file descriptor returned by the io.open () function. The file must be opened in read mode.
- **noBytes** (integer) – number of bytes to read. The function may read less bytes than requested when reaching the end of file.

**Return value**

- (string) – a string with noBytes (or less) characters.
- "" – empty string after reaching the end of file.

**Examples**

```lua
local appName = "Test 26 - IO Access"

local function init()
    -- Read file
    local val
    local f = io.open("Foo.txt","r")
    if(f) then
        local data = io.read(f, 10)
        io.close(f)
        val = tonumber(data)
    end

    -- Increment the counter
    if(not val) then
        val=1
    else
        val=val+1
    end

    -- Print to console
    print("New value: ", val)

    -- Write back
    f = io.open("Foo.txt","w")
    if(f) then
        io.write(f, val,"\n")
        io.close(f)
    end
end

return {init=init, author="JETI model", version="1.00", name=appName}
```

![Console output](image)
**io.readall (<path>)**

*(since V4.22)*

The readall () function reads the whole contents of the file with specified path.

**Parameters**

- **path** – string that specifies path to the file.

**Return value**

- (string) – a string containing the contents of the file.
- nil – if error occurred.

**Examples**

```lua
-- Prints the contents of Text.txt
local file = io.readall("Text.txt")
print(file)
```
**io.write (<fileObject>, <data>[, <data>, ...])**

The write () function writes *data* to an opened file.

**Parameters**
- **fileObject** – a file descriptor returned by the io.open () function. The file must be opened in *write* or *append* mode.
- **data** – any Lua type that can be converted into string. If more than one data parameter is used their contents are written to the file by one in the same order as they are specified.

**Return value**
- **fileObject** – if data was successfully written.
- **nil, <errorString>, <errorNumber>** - if the data can't be written.

**Examples**
See `io.read (<fileObject>, <noBytes>)`
io.seek (<fileObject>, <offset>)

The seek () function moves the current file pointer to a new absolute position.

Parameters

- **fileObject** – a file descriptor returned by the io.open () function.
- **offset** (integer) – new absolute position from the beginning of the file. It must be a positive integer. If specified offset is bigger than the file size, the pointer is moved to the end of the file.

Return value

- (integer) – “0” in case of success, other values mean failure.

Examples

```lua
local appName = "Test 27 - IO Seek"
--------------------------------------------------------------------------------
local function init()
    f = io.open("Foo-2.txt","w")
    if(f) then
        io.write(f, "Hello World!")
        io.seek(f,6)
        io.write(f, "JETI DC-24!")
        io.close(f)
        -- Foo-2 now contains "Hello JETI DC-24!"
    end
end
--------------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.00", name=appName}
```
**json.encode (<value>)**


The encode () function serialises a Lua value into a string containing the JSON representation.

Standards compliant JSON must be encapsulated in either an object ({} ) or an array ([ ]). If strictly standards compliant JSON is desired, a table must be passed to json.encode function.

**Parameters**
- **value** – can be one of these types: boolean, lightuserdata (NULL only), nil, number, string, table. Other types generate a runtime error.

**Return value**
- (string) – a string representation of the value.

**Examples**

```lua
value = { true, { foo = "bar" } }
json_text = json.encode(value)
-- Returns: '[true,{"foo":"bar"}]'
```
The `json_decode` function deserialises any UTF-8 JSON string into a Lua value or table. The function requires that any NULL (ASCII 0) and double quote (ASCII 34) characters are escaped within strings. All escape codes will be decoded and other bytes will be passed transparently. UTF-8 characters are not validated during decoding and should be checked elsewhere if required.

JSON null will be converted to a NULL userdata value. Numbers incompatible with the JSON specification (infinity, NaN, hexadecimal) can be decoded as well.

Care must be taken after decoding JSON objects with numeric keys. Each numeric key will be stored as a Lua string. Any subsequent code assuming type number may break.

**Parameters**
- **text** – a string representation of the value.

**Return value**
- A decoded value.

**Examples**

```lua
json_text = '[ true, { "foo": "bar" } ]'
value = json_decode(json_text)
-- Returns: { true, { foo = "bar" } }
```
**system.getCPU ()**

This function returns current system instructions usage by the Lua interpreter. Using this function you can prevent enormous CPU usage. If the script reaches 100% utilization, it will be killed. The CPU usage counter resets before the transmitter calls any Lua function (init, loop ...).

**Parameters**

- none

**Return value**

- (number) – integer range from 0 to 100, meaning 0 as the least CPU utilization.

**Examples**

```lua
-- The script is killed after reaching 100% CPU utilization
local function init()
    while true do
        -- A dummy cycle
        for val = 1, 100 do
            end
            print ( system.getCPU() )
        end
    end
end
return {init=init, author="JETI model", version="1.0"}
```
**system.getTime ()**
The function gets the current time as seconds elapsed since January 1, 2000, 00:00:00.

**Parameters**

none

**Return value**

- (number) – current time. Please note that the numeric representation uses 32-bit signed integer which causes possible overflow in 2068.

**Examples**

```lua
-- Displays a blinking text
local function printForm()
    if(system.getTime() % 2 == 0) then
        lcd.drawText(10,30,"Blinking text",FONT_MAXI)
    end
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 4 - Get Time",nil, nil,printForm)
end

return {init=init,author="JETI model", version="1.0"}
```

![Blinking text example](image-url)
**system.getTimeCounter()**

The function gets current timestamp in milliseconds. The timestamp counts from zero (cleared after the transmitter resets). This function can be used to measure small time increments between the loop calls.

**Parameters**

none

**Return value**

- (number) – current timestamp. Please note that the numeric representation uses 32-bit signed integer.

**Examples**

```lua
-- Prints the average period between calls of the Loop function
local lastTime
local avgTime
-- Calculates the average period
local function loop()
  local newTime = system.getTimeCounter()
  local delta = newTime - lastTime
  lastTime = newTime
  if (avgTime == 0) then
    avgTime = delta
  else
    avgTime = avgTime * 0.95 + delta * 0.05
  end
end
-- Displays an average time between loops
local function printForm()
  lcd.drawText(10,30,string.format("Avg. time: %.2fms",avgTime),FONT_MAXI)
end

local function init()
  system.registerForm(1,MENU_MAIN,"Test 5 - Avg. time",nil, nil,printForm)
  lastTime = system.getTimeCounter()
  avgTime = 0
end

return {init=init, loop=loop, author="JETI model", version="1.0"}
```

![Screenshot of the average time calculation example](image.png)
**system.getTime()**

This function retrieves the current date and time as a table.

**Parameters**

none

**Return value**

- (table) with the following elements:
  - "year" (2000+),
  - "mon" (1-12),
  - "day" (1-31),
  - "hour" (0-24),
  - "min" (0-59),
  - "sec" (0-59),
  - "dst" (true/false for daylight saving time, introduced in V4.22).

**Examples**

-- Prints the current date and time

```lua
local function init()
    local dt = system.getTime()
    print (string.format("Time: %d-%02d-%02d, %d:%02d:%02d",
        dt.year, dt.mon, dt.day, dt.hour, dt.min, dt.sec))
end
```

```
return {init=init, author="JETI model", version="1.0"}
```
system.getVersion()

This function gets the transmitter SW version (e. g. “4.00”).

Parameters

none

Return value

- (string) – Tx version.

Examples
**system.getDeviceType ()**

This function gets the device type as string (e.g. “JETI DC-24”).

**Parameters**

none

**Return value**

- (string), (integer) – Device type, emulator flag (1 – script running inside emulator, 0 – physical device).

**Examples**
system.getLocale ()
This function gets the current system locale (e. g. “en”). It is useful if you want to translate the application into several languages.

Parameters
none

Return value
- (string) – locale (cz, de, en, fr, it, pt...).

Examples

```lua
-- Displays localized texts
local translations = {en = "Voltage", cz = "Napětí", de = "Spannung", fr = "Tension"}

local function printForm()
    local locale = system.getLocale()
    -- Use English as default locale
    local voltage = translations[locale] or translations["en"]
    lcd.drawText(10,10,voltage .. ": 10V")
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 7 - Localization",nil, nil,printForm)
end

return {init=init, author="JETI model", version="1.0"}
```
system.getTxTelemetry ()

This function retrieves the system voltage, signal quality etc.

Parameters
none

Return value
- (table) - table containing basic transmitter and receiver telemetry data:
  - "txVoltage" – transmitter voltage [V]
  - "txBattPercent" – remaining battery capacity in percent
  - "txCurrent" – transmitter current [mA]
  - "txCapacity" – battery capacity (charged/discharged) [mAh]
  - "rx1Percent" – signal quality of the primary receiver [%]
  - "rx1Voltage" – primary receiver voltage [V]
  - "rx2Percent" – signal quality of the secondary receiver [%]
  - "rx2Voltage" – secondary receiver voltage [V]
  - "rx BVoltage" – backup (900MHz) receiver voltage [V]
  - "rxBPercent" – signal quality of the backup receiver [%]
  - "photoValue" – raw value of the light intensity sensor (range from 0 to 4096)
  - "RSSI" – a signal strength indicator (table). Higher value means stronger signal. The table values are specified in the following order:
    - Antenna 1 of Rx1,
    - Antenna 2 of Rx1,
    - Antenna 1 of Rx2,
    - Antenna 2 of Rx2,
    - Antenna 1 of backup Rx,
    - Antenna 2 of backup Rx.

Note: Some of the telemetry values are not available in DC/DS-16.

Examples
-- Displays Rx/Tx telemetry
local txTel = system.getTxTelemetry();

lcd.drawText(10,20, string.format("Rx1: %dV, Q=%d%%, A1/2=%d/%d", txTel.rx1Voltage, txTel.rx1Percent, txTel.RSSI[1], txTel.RSSI[2]))

lcd.drawText(10,40, string.format("Rx2: %dV, Q=%d%%, A1/2=%d/%d", txTel.rx2Voltage, txTel.rx2Percent, txTel.RSSI[3], txTel.RSSI[4]))

lcd.drawText(10,60, string.format("Rx B: %dV, Q=%d%%, A1/2=%d/%d", txTel.rxBVoltage, txTel.rxBPercent, txTel.RSSI[5], txTel.RSSI[6]))

lcd.drawText(10,80, string.format("Tx: %.2fV, Batt=%d%%, I=%.2fmA", txTel.txVoltage, txTel.txBattPercent, txTel.txCurrent))
**system.getUserN\text{a}me ()**

This function retrieves the user name (can be set in System – Configuration).

**Parameters**
- none

**Return value**
- (string) - user name

**Examples**
system.getSerialCode ()
This function retrieves the transmitter registration code (see the System – Installed Modules menu).

Parameters
none

Return value
• (string) - registration code in the following format: “XXXX-XXXX-XXXX-XXXX”

Examples
system.getSensors ()

This function retrieves all detected sensors/values in a table. Please refer to the JETI Telemetry Communication Protocol for further details on available data types and EX telemetry format:


Parameters
none

Return value
- (table) - a list of telemetry entries. Each entry contains the following parameters:
  - "id" (integer) – sensor unique identifier.
  - "param" (integer) – telemetry parameter identifier (0 stands for sensor label).
  - "decimals" (integer) – number of digits after the decimal point.
  - "type" (integer) – telemetry data type (e.g. 5 = date/time, 9 = GPS coordinates).
  - "label" (string) – telemetry label (or name of the sensor).
  - "unit" (string) – sensor unit. Only default units are available, without conversion. Meters for distance, meters per second for speed etc.
  - "valid" (boolean) – true if the telemetry value has been refreshed recently.
  - "valSec", "valMin", "valHour" (integer) – value representation for the “time” data type.
  - "valYear", "valMonth", "valDay" (integer) – value representation for the “date” data type.
  - "value", "min", "max" (float) – value representation for all numerical data types.
  - "valGPS" (integer-coded) – value representation for the GPS coordinates.
Examples

-- Displays all available sensors and their values

local function init()
    local sensors = system.getSensors()
    for i, sensor in ipairs(sensors) do
        if (sensor.type == 5) then
            if (sensor.decimals == 0) then
                -- Time
                print(string.format("%s = %d:%02d:%02d", sensor.label, sensor.valHour, sensor.valMin, sensor.valSec))
            else
                -- Date
                print(string.format("%s = %d-%02d-%02d", sensor.label, sensor.valYear, sensor.valMonth, sensor.valDay))
            end
        elseif (sensor.type == 9) then
            -- GPS coordinates
            local nesw = {"N", "E", "S", "W"}
            local minutes = (sensor.valGPS & 0xFFFF) * 0.001
            local degs = (sensor.valGPS >> 16) & 0xFF
            print(string.format("%s = %d° %f' %s", sensor.label, degs, minutes, nesw[sensor.decimals+1]))
        else
            if(sensor.param == 0) then
                -- Sensor label
                print (string.format("%s:",sensor.label))
            else
                -- Other numeric value
                print (string.format("%s = %.1f %s (min: %.1f, max: %.1f)", sensor.label, sensor.value, sensor.unit, sensor.min, sensor.max))
            end
        end
    end
end

return {init=init, author="JETI model", version="1.0"}
**system.getSensorByID (<sensor ID>, <sensor param>)**
This function gets a single sensor value based on sensor ID and a specified parameter.

**Parameters**
- **Sensor ID** – a unique identifier of the requested sensor.
- **Sensor param** – requested sensor parameter (0 stands for sensor label).

**Return value**
- (table) – telemetry entry as specified in system.getSensors ()
- nil – if the entry doesn’t exist

**Examples**
system.getInputs (<Input 1>[, <Input 2>][, <Input 3>], ...)

This function gets multiple values of sticks/switches. Up to 8 inputs can be defined. The function guarantees that all values are retrieved at the same time/Tx frame. The function call is protected by a mutex so values will not change within the function call.

Parameters

- Input 1-n (string) – specified input controls.

<table>
<thead>
<tr>
<th>Allowed controls</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sticks/proportional controls</td>
<td>P1, P2, P3, P4, P5, P6, P7, P8</td>
</tr>
<tr>
<td>Switches</td>
<td>SA, SB, SC, SD, SE, SF, SG, SH, SI, SJ, SK, SL</td>
</tr>
<tr>
<td>Trainer inputs (wireless trainer)</td>
<td>T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16</td>
</tr>
<tr>
<td>PPM inputs</td>
<td>CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8</td>
</tr>
<tr>
<td>Servo outputs</td>
<td>O1, O2, ..., O24</td>
</tr>
</tbody>
</table>

Return value

- (list) – a list of input values within range <-1, 1>. If the control doesn’t exist, nil will be returned.

Examples

```
local sa, sb, p1, p2 = system.getInputs("SA","SB","P1","P2")
-- sa, sb, p1, p2 now contain numbers within range <-1, 1>
```
system.getInputsVal (<Input 1>[, <Input 2> [, <Input 3>, ...])

This function gets the current input value based on a "SwitchItem" datatype. Up to 8 switches can be defined. The function guarantees that all values are retrieved at the same time/frame. It allows users to select any proportional control as an input. The function call is protected by a mutex so values will not change within the function call.

Parameters
- Input 1-8 (SwitchItem or nil) – specified input controls.

Return value
- (list) – list of input values within range -1, 1>. If the control doesn’t exist, nil will be returned.

Examples
-- Assigned switch as a local variable
local switch
-- Form initialization
local function initForm()
    form.addRow(2)
    form.addLabel({label="Selected switch"})
    form.getInputbox(switch, true, function(value) switch = value end)
end
-- Print the value if the switch is assigned
local function printForm()
    local val = system.getInputsVal(switch)
    if(val) then
        lcd.drawNumber (10, 50, val * 100, FONT_MAXI)
    end
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 9 - Input form",initForm, nil,printForm)
end
--------------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.0"}
system.getIMU ([<smoothed = 0>]) – Only DS
This function returns the current attitude calculated from accelerometer/gyro inputs. This is not an Euler attitude representation since the rotations aren't consecutive rotations but only angles between Earth and the IMU.

Parameters
- **smoothed** (integer) – optional parameter that says the attitude should be returned after all the transmitter processing has been applied (filtering, smoothing, rates and pitch offset). If this parameter is zero (by default), no additional filtering is applied.

Return value
- (table) – a table consisting of the following keys:
  - r – “roll axis”. Angle between the Earth ground plane and the IMU Y axis.
  - p – “pitch axis”. Angle between the Earth ground plane and the IMU X axis.
  - y – “yaw axis”. Angle between the Earth North and the IMU X axis.
  - ax – normalized real device acceleration adjusted to remove gravity (1G ~ 1.0).
  - ay – normalized real device acceleration adjusted to remove gravity (1G ~ 1.0).
  - az – normalized real device acceleration adjusted to remove gravity (1G ~ 1.0).

Examples
**system.getRawIMU() – Only DS**

Returns raw IMU (Inertial Motion Unit) data. The data format is 16-bit signed so the range values could be anything between -32768 and 32767.

**Parameters**
None

**Return value**
- (table) – a table consisting of the following keys:
  - ax – raw accelerometer data for X axis. Maximum range is \( \pm 2 \text{G} \).
  - ay – raw accelerometer data for Y axis. Maximum range is \( \pm 2 \text{G} \).
  - az – raw accelerometer data for Z axis. Maximum range is \( \pm 2 \text{G} \).
  - gx – raw gyroscope data for X (pitch) axis. Maximum range is \( \pm 2000^\circ/\text{s} \).
  - gy – raw gyroscope data for Y (roll) axis. Maximum range is \( \pm 2000^\circ/\text{s} \).
  - gz – raw gyroscope data for Z (yaw) axis. Maximum range is \( \pm 2000^\circ/\text{s} \).

**Examples**
**system.setProperty** (<property_name>, <value>)

Can set some of the system properties.

**Parameters**
- **property_name** (string) – name of the property.
- **value** – value to be set.

<table>
<thead>
<tr>
<th>Property</th>
<th>Possible values</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>WirelessMode</td>
<td>Teacher, Student, Default</td>
<td></td>
</tr>
<tr>
<td>WirelessEnabled</td>
<td>0, 1</td>
<td>Enables/disables wireless system.</td>
</tr>
<tr>
<td>Volume</td>
<td>0-16</td>
<td>Sets the system volume.</td>
</tr>
<tr>
<td>VolumePlayback</td>
<td>0-100</td>
<td>Sets the audiofile-playback volume.</td>
</tr>
<tr>
<td>VolumeBeep</td>
<td>0-100</td>
<td>Sets the beep volume</td>
</tr>
<tr>
<td>Backlight</td>
<td>0-1000</td>
<td>Sets the backlight value</td>
</tr>
<tr>
<td>Color</td>
<td>0-11</td>
<td>Sets the system color profile by index (only DC/DS-24)</td>
</tr>
<tr>
<td>BacklightMode</td>
<td>0 (Off) – 3 (Always)</td>
<td>Sets the backlight mode</td>
</tr>
</tbody>
</table>

**Return value**
none

**Examples**

```lua
-- sets the wireless teacher mode (doesn't work in Double Path).
system.setProperty("WirelessMode", "Teacher")

-- sets the wireless student mode (doesn't work in Double Path).
system.setProperty("WirelessMode", "Student")

-- sets the wireless mode to default (doesn't work in Double Path).
system.setProperty("WirelessMode", "Default")

-- Enables (1) or disables (0) wireless transmission
system.setProperty("WirelessEnabled", 1)
```
**system.getProperty (<propertyName>)**

Since V4.20

Can retrieve some of the system properties by name.

**Parameters**

- **propertyName** (string) – name of the property.

**Return value**

- **value** – value of the property.

<table>
<thead>
<tr>
<th>Property</th>
<th>Return values</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>WirelessMode</td>
<td>Teacher, Student, Default</td>
<td></td>
</tr>
<tr>
<td>WirelessEnabled</td>
<td>0, 1</td>
<td></td>
</tr>
<tr>
<td>Volume</td>
<td>0 - 16</td>
<td></td>
</tr>
<tr>
<td>VolumePlayback</td>
<td>0 - 100</td>
<td></td>
</tr>
<tr>
<td>VolumeBeep</td>
<td>0 - 100</td>
<td></td>
</tr>
<tr>
<td>Backlight</td>
<td>0 - 1000</td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>0 - 11</td>
<td>Only DC/DS-24</td>
</tr>
<tr>
<td>BacklightMode</td>
<td>0 (Off) – 3 (Always)</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>String</td>
<td>Returns the model name</td>
</tr>
<tr>
<td>ModelFile</td>
<td>String</td>
<td>Returns the filename of current model.</td>
</tr>
</tbody>
</table>

**Examples**
system.getSwitchInfo (<switch>)

Since V4.22

Can retrieve parameters of the assigned switch (given a SwitchItem data type).

Parameters

- **switch** (SwitchItem) – a switch that shall be investigated.

Return value

- **table** – properties of the switch object:
  - **label** (string) – the switch label (example: P1 - P8 for proportional controls, Sa – Sl for switches…).
  - **value** (number) – current switch value in the range (-1, 1).
  - **proportional** (boolean) – true if the assigned control is evaluated proportionally.
  - **assigned** (boolean) – true if the control is assigned.
- **nil** – in case of error.
system.messageBox (<message>, <timeout = 3>)

After calling the messageBox () function, the status message will be shown for a short period specified by the Timeout parameter.

Parameters
- **Message** (string) – message to be displayed.
- **Timeout** – display timeout specified in seconds.

Return value
none

Examples

```lua
system.messageBox("Warning, low altitude", 5)
system.messageBox("", 0)  -- clears the message immediately
```
**system.registerTelemetry** ( <windowNo>, <label>, <size>, <printFunction> )

The registerTelemetry () function registers a new telemetry window that can be displayed on the main screen (Desktop). The telemetry window is then accessible in the Timers/Sensors – Displayed Telemetry menu. Up to two independent telemetry windows can be created per application. Lua telemetry windows on the Desktop will be included in the model configuration file.

Recommended is to call this function during the application initialization.

**Parameters**
- **windowNo** (integer) – window identifier (either 1 or 2; max. 2 windows are allowed).
- **label** (string, max. 31Bytes long) – window label.
- **size** (integer) – size of the telemetry window, can be one of the following values:
  - 0 – automatic size (the window sill occupy either 1 or 2 positions as selected by the user)
  - 1 – only small window is allowed, the window will occupy only one position.
  - 2 – only big window is allowed (2 positions).
  - 3 – the whole screen is reserved for the Lua telemetry (keeping status bar visible at the bottom left).
  - other – the whole screen is reserved for the Lua telemetry (without the status bar at the bottom left).
- **printFunction** – a function that is called to draw the contents of the telemetry window. It takes two parameters: width and height of the drawing canvas. The canvas is always cleared before calling the printFunction.

**Return value**
- (integer) – registered number (windowNo) on success.
- nil – in case of error.

**Examples**

```lua
-- Displays the telemetry window width and height
local function printDimensions(width, height)
    lcd.drawText(5,5,width.." x " .. height.."px")
end

local function init()
    -- registers a single-position-only window
    system.registerTelemetry(1,"Lua small window",1,printDimensions)
    -- registers a double-position-only window
    system.registerTelemetry(2,"Lua large window",2,printDimensions)
end
```

```
return {init=init, author="JETI model", version="1.0"}
```
After selecting the appropriate Lua telemetry windows, the transmitter will treat them as standard displayed telemetry:
**system.unregisterTelemetry ( <windowNo> )**

The unregisterTelemetry () function disables the registered Lua window.

**Parameters**
- **windowNo** (integer) – window identifier (either 1 or 2).

**Return value**
- none

**Examples**

```
    system.unregisterTelemetry(1)
```
system.registerForm( <formNo>,<parentMenuID>, <label>,
<initFunction>, <keyPressFunction>, <printFunction> )

The registerForm () function registers a new interactive form that can be placed in one of the transmitter menus. Up to two independent Lua forms can be created per application. The form can contain up to 127 subforms managed manually by the application programmer.

Recommended is to call this function during the application initialization.

Parameters

- formNo (integer) – form identifier (either 1 or 2; max. 2 windows are allowed).
- parentMenuID - one of the Menu definition: MENU_MAIN, MENU_FINE, MENU_ADVANCED ... (or 0 if the form has to be shown immediately)
- label (string, max. 31Bytes long) – form label.
- initFunction – a function called after the form is created. The initFunction takes a single parameter, a subform ID (1 by default). Its purpose is to create all necessary components using the form library.
- keyPressFunction – a function called every time the button is pressed or released. It takes a single parameter, keyCode.
- printFunction – a function called to draw the contents of the form window. It takes zero parameters. The form canvas is always cleared and filled with background before calling the printFunction.

Return value

- (integer) – registered number (formNo) on success.
- nil – in case of error.
Examples

```
-- Current key
local key = 0

-- Form initialization
local function initForm(subform)
    form.addLabel({label="Subform ".subform,font=FONT_BIG})
    if(subform == 1) then
        -- Link to the following subform
        form.addLink((function() form.reinit(2) end), {label = "Next >>"})
    else
        -- Link to the first subform
        form.addLink((function() form.reinit(1) end), {label = "<< Back"})
    end
end

-- Latches the current keyCode
local function keyForm(keyCode)
    if(keyCode == KEY_RELEASED) then
        key = 0
    else
        key = keyCode
    end
end

-- Prints the last button pressed
local function printForm()
    lcd.drawText(10,80,"Pressed: ..key, FONT_MAXI)
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 10 - Form",initForm, keyForm,printForm)
end

--------------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.0"}
```
system.unregisterForm( <formNo> )

The unregisterForm() function disables the registered Lua form.

Parameters

- formNo (integer) – window identifier (either 1 or 2).

Return value

none

Examples

system.unregisterForm(1)
system.registerControl ( <controlNo>, <label>, <shortLabel>)

The registerControl () function registers a new output control that can be assigned to any model function using a standard Input selection dialogue. Up to 10 controls can be created per model (a single application with 10 outputs or 10 applications, each with a single output).

Recommended is to call this function during the application initialization.

Parameters
- controlNo (integer) – desired control identifier (from 1 to 10; max. 10 controls are allowed per model).
- label (string, max. 31Bytes long) – control label.
- shortLabel (string, max. 3Bytes long) – short identifier.

Return value
- (integer) – registered number (controlNo) on success.
- nil – in case of error.

Examples
See system.setControl ( <controlNo>, <value>, <delayms>[, <smoothType = 0>])
system.setControl ( <controlNo>, <value>, <delayms>[, <smoothType = 0>])

The setControl () function sets the value of a registered control. Prior to calling this function the control must be successfully registered using system.registerControl ( <controlNo>, <label>, <shortLabel>). You can set the value only to controls registered inside the same application where system.registerControl ( <controlNo>, <label>, <shortLabel>) has been called.

Parameters
- controlNo (integer) – desired control identifier (from 1 to 10; max. 10 controls are allowed per model).
- value (float) – a new value within range <-1, 1>.
- delayms (integer) – A transition delay to be applied (or smoothing intensity). If the delay equals 0, the value is set immediately.
- smoothType (integer) – smoothing type. “0” means linear interpolation, “1” represents a lowpass filtering algorithm.

Return value
- true – on success.
- nil – in case of error.

Examples

```lua
local appName="Test 28 - Controls"
local ctrlIdx
--------------------------------------------------------------------
-- Loop function
local function loop()
    -- low pass filtered control based on P2 value
    if(ctrlIdx) then
        system.setControl(1, system.getInputs("P2") ,1000,1)
    end
end
--------------------------------------------------------------------
-- Init function
local function init()
    lastTimeChecked = system.getTimeCounter()
    ctrlIdx = system.registerControl(1, "LowPass Ctrl","C01")
end
--------------------------------------------------------------------
return { init=init, loop=loop, author="JETI model", version="1.00",name=appName}
```
**system.pLoad ( <param> [, <defaultValue>] )**

The `pLoad()` function loads a persistent parameter from model configuration. The model configuration is loaded every time you turn on your transmitter or after you switch to another model. This happens before calling the `init()` function so you can load all necessary parameters right in the `init()` function. You can call the `system.pLoad()` function at any time.

There are, however, several restrictions that apply to saving and loading model-specific parameters. You should not use more than **30 parameters** per application since this can result in large system memory consumption and slow loading times while parsing the model data.

You can serialize the following data types:

- (string) – a sequence of characters. The sequence must be shorter than **64 Bytes** and must not contain any nonprintable characters (ASCII code must be greater than 31). UTF-8 characters are supported but please note that the number of characters is not equal to the string Byte size.
- (integer) – an integer value, 32bit long.
- (SwitchItem) – a switch definition, see `system.getInputsVal (<Input 1>, <Input 2>, <Input 3>, ...)` function.
- nil – useful to delete a persistent parameter.

**Parameters**

- **param** (string) – parameter identifier. Must be shorter than 64 Bytes and must not contain any nonprintable characters.
- **defaultValue** – a default value if the parameter is not found. It can be any of the Lua supported data types.

**Return value**

- Loaded value (string, integer or SwitchItem) in case of success.
- `<defaultValue>` or nil in case of failure
Examples

```lua
local appName = "Test 20 - Load/Save"
local switch, number, text
-- Function callbacks
local function textChanged(value)
    text=value
    system.pSave("text",value)
end

local function numberChanged(value)
    number=value
    system.pSave("number",value)
end

local function switchChanged(value)
    switch=value
    system.pSave("switch",value)
end
-- Form initialization
local function initForm(subform)
    form.addRow(2)
    form.addLabel({label="Text"})
    form.addTextbox(text,20,textChanged)

    form.addRow(2)
    form.addLabel({label="Number"})
    form.addIntbox(number,0,100,0,0,1,numberChanged)

    form.addRow(2)
    form.addLabel({label="Switch"})
    form.addInputbox(switch,true,switchChanged)
end
-- Init
local function init()
    system.registerForm(1,MENU_MAIN,appName,initForm)
    text = system.pLoad("text","Foo")
    switch = system.pLoad("switch")
    number = system.pLoad("number",10)
end

return { init:init, author="JETI model", version="1.0", name=appName}  
```
**system.pSave ( <param>, <value> )**

The `pSave()` function saves a persistent parameter to model configuration. The parameter is not saved immediately but when you switch to another model or turn off the transmitter. You can call the `system.pSave()` function at any time.

**Parameters**

- **param** (string) – parameter identifier. Must be shorter than 64 Bytes and must not contain any nonprintable characters.
- **value** - a new value. Can be one of the following types:
  - (string) – a sequence of characters. The sequence must be shorter than 64 Bytes and must not contain any nonprintable characters (ASCII code must be greater than 31). UTF-8 characters are supported but please note that the number of characters is not equal to the string Byte size.
  - (integer) – an integer value, 32bit long.
  - (SwitchItem) – a switch definition, see `system.getInputsVal ( <Input 1>, <Input 2>, <Input 3>, ... )` function.
  - (table) – must be a “basic” array-type table consisting of only integers and strings. Maximum number of elements in the table is limited to 32. Table keys are not kept and must be integer.
  - nil – useful to delete a persistent parameter.

**Return value**

- true in case of success.
- nil otherwise

**Examples**

```lua
-- Save the contents of an array
system.pSave("array", {1,2,3,"Text 1", "Text 2"})
```

See `system.pLoad ( <param> [, <defaultValue>] )`
system.vibration ( <leftRight>, <vibrationProfile> )

Starts vibration using one of the predefined schemes.

**Parameters**
- **leftRight** (boolean) – selects left (false) or right (true) stick.
- **vibrationProfile** (integer) – sets the vibration profile according to the following constants:
  - 1 – long pulse
  - 2 – short pulse
  - 3 – 2 × short pulse
  - 4 – 3 × short pulse
  - Other – stops vibration

**Return value**
none

Note: not supported on DC/DS-16.

**Examples**

```lua
-- Starts vibration (two short pulses) for both sticks
local left = false
system.vibration(left, 3);
system.vibration(not left, 3);
```
system.playFile (<filename>, <playbackType>)

The playFile() function plays a given file from the SD card. If the path is absolute ("/someaudio.wav"), an absolute filepath will be selected. Otherwise, the transmitter will start looking for the given file inside "/Audio" or "/Audio/XX" folders (XX stands for language abbreviation).

Parameters
- **filename** (string) – absolute or relative path to the file.
- **playbackType** (integer) – one of the playback types from the Audio definitions section:
  - AUDIO_BACKGROUND - playback starts immediately in the background (playback is not interruptible by alarms). Background playback accepts WAV and MP3 files.
  - AUDIO_IMMEDIATE - playback starts immediately in the foreground queue (playback can be interrupted by alarms). The foreground playback accepts only WAV files.
  - AUDIO_QUEUE - playback starts as soon as the foreground queue is empty (playback can be interrupted by alarms). The foreground playback accepts only WAV files. Default option.

Return value
- none

Examples
See form.addAudioFilebox (<currentAudioFile>, <changedCallback>, <paramTable>)

```
-- plays "filename.mp3" inside the "Audio" folder. Stops any previous background playback. MP3 files can be played only in the background.
system.playFile("filename.mp3", AUDIO_BACKGROUND)

-- plays "filename.wav" inside the "Audio" folder immediately. Stops any previous foreground playback.
system.playFile("filename.wav", AUDIO_IMMEDIATE)

-- plays "filename.wav" in the root folder. The file is added to the end of the queue.
system.playFile("/filename.wav", AUDIO_QUEUE)
```
system.playNumber( <value>, <decimals> [, <unit>, <label>] )

The playNumber() function activates the text-to-speech algorithms to announce the numerical value by voice. The voice announcements are always appended to the foreground audio queue. The voice output is always processed according to the selected language.

Parameters
- **value** (float) – number to be played.
- **decimals** (integer) – playback precision. Number of decimals can be set from 0 to 2.
- **unit** (string) – optional unit to be announced. The unit must correspond to the units specified in “Voice/XX/numbers.jsn” file. See the Supported units.
- **label** (string) – optional label. The label must correspond to the labels specified in “Voice/XX/numbers.jsn” file. See the Supported labels.

Return value
- true – in case of success.
- nil – otherwise.

Supported units

<table>
<thead>
<tr>
<th>Wmi, F, °C, °, W, s, min, h, mAh, Ah, A, V, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>hPa, kPa, psi, atm, b</td>
</tr>
<tr>
<td>m/s, ft./s, km/h, kt., mph</td>
</tr>
<tr>
<td>m, ft, km, mi., yd.</td>
</tr>
<tr>
<td>ml, l, hl, floz, gal</td>
</tr>
<tr>
<td>ml/m, l/m, oz/m, gpm</td>
</tr>
</tbody>
</table>

Supported English labels

<table>
<thead>
<tr>
<th>Voltage, Current, Run time, U Rx, A1, A2, T, Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell 1, Cell 2, Cell 3, Cell 4, Cell 5, Cell 6, LowestVolt, LowestCell, Accu. volt</td>
</tr>
<tr>
<td>Vario, Abs. altit, Rel. altit, Air press.</td>
</tr>
<tr>
<td>U Battery, I Battery, U BEC, I BEC, Capacity, Revolution, Temp., Run Time, PWM</td>
</tr>
<tr>
<td>Quality, SatCount, Altitude, AltRelat., Distance, Course, Azimuth, Impulse, Trip</td>
</tr>
<tr>
<td>R.volume, R.volumeP, Flow, Pressure</td>
</tr>
</tbody>
</table>

Examples
- plays "Altitude: one-hundred-twenty-fiwe-point-one meters"
  system.playNumber (125.1, 1, "m", "Altitude")
- plays "One Volt"
  system.playNumber (1.2345, 0, "V")
- plays "One-point-two-three Volts"
  system.playNumber (1.2345, 2, "V")
**system.playBeep( <repeatCount>, <frequency>, <length> )**

The `playBeep()` function performs an audible beep.

**Parameters**
- `repeatCount` (integer) – Number of successive beeps after the first one (0 – 10).
- `frequency` (integer) – frequency [Hz] (accepted values between 200 – 10000).
- `length` (integer) – beep length [ms] (accepted values 20 – 10000).

**Return value**
- none

**Examples**
- Beeps 3-times with frequency 5kHz and beep length 100ms.
  `system.playBeep (2, 5000, 100)`
- Beeps once with frequency 4kHz and beep length 20ms
  `system.playBeep (0, 4000, 20)`
**system. playSystemSound ( <soundIndex>)**

The `playSystemSound()` function plays a given system audio file (if it has been already assigned in the **System – System Sounds** menu).

**Parameters**
- **soundIndex** (integer) – Index of the system sound. Can be one of the [Audio definitions]:
  - SOUND_START, SOUND_BOUND, SOUND_LOWTXVOLT, SOUND_LOWSIGNAL, SOUND_SIGNALLOSS, SOUND RANGE TEST, SOUND_AUTOTRIM, SOUND_INACT, SOUND_LOWQ.

**Return value**
- none

**Examples**
- Plays "Low signal quality" alarm
  system.playSystemSound (SOUND_LOWQ)
- Plays "System startup" audiofile
  system.playSystemSound (SOUND_START)
**system. isPlayback ()**

The isPlayback () function checks if any audio file is being played.

**Parameters**

none

**Return value**

- true – if any audio queue (background or foreground) is not empty and an audio file is being played.
- false – otherwise.

**Examples**

```lua
-- Continuously announces the transmitter runtime in whole seconds.
local function loop()
    if(not system.isPlayback()) then
        local newTime = system.getTimeCounter()
        system.playNumber(newTime/1000,0,"s","T")
    end
end

local function init()
end

return {init=init, loop=loop, author="JETI model", version="1.0"}
```
system. stopPlayback ([<playbackType>])

The stopPlayback() function stops the audio playback and optionally clears the audio queue. If the playbackType is not specified, both foreground and background playback are stopped and the foreground audio queue is cleared.

Parameters

- **playbackType** – if specified, it can be one of the following values:
  - AUDIO_BACKGROUND – the function stops only background playback.
  - AUDIO_IMMEDIATE – the function stops only foreground playback and clears the foreground queue.

Return value

none

Examples

-- Checks "Sa" to stop the current playback completely
local function loop()
  local sa = system.getInputs("SA")
  if(sa > 0) then
    system.stopPlayback()
    -- After the playback is stopped, any urgent warning/voice telemetry can
    -- be played.
  end
end

local function init()
end

return {init=init, loop=loop, author="JETI model", version="1.0"}
**lcd.setColor (<red>, <green>, <blue>[,<alpha = 255])**

The `setColor()` function sets the current LCD color for further drawing.

The `lcd draw` functions can be called only from the functions with explicit LCD support (e. g. form and telemetry `printFunction`).

**Parameters**

- **red** (integer) – red value (0 – 255).
- **green** (integer) – green value (0 – 255).
- **blue** (integer) – blue value (0 – 255).
- **alpha** (integer) – transparency value (0 – 255). “0” for full transparency.

**Return value**

none

Note: The function has no effect on DC/DS-16.

**Examples**

-- Prints several colored rectangles

```lua
local level = 200

local function printForm()
    lcd.setColor(level,0,0)
    lcd.drawFilledRectangle(10,10,50,50)
    lcd.setColor(0,level,0)
    lcd.drawFilledRectangle(60,60,50,50)
    lcd.setColor(0,0,level)
    lcd.drawFilledRectangle(110,10,50,50)
    lcd.setColor(0,0,level)
    lcd.drawFilledRectangle(60,10,50,50)
    lcd.setColor(level,level,0)
    lcd.drawFilledRectangle(160,60,50,50)
    lcd.setColor(0,level,level)
    lcd.drawFilledRectangle(210,10,50,50)
    lcd.setColor(0,level,level)
    lcd.drawFilledRectangle(260,60,50,50)
    -- Semi-transparent rectangle
    lcd.setColor(0,0,0,50)
    lcd.drawFilledRectangle(20,45,280, 30)
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 12 - Drawing",nil, nil,printForm)
end

return {init=init, author="JETI model", version="1.0"}"
```
**lcd.drawPoint (<x>, <y>)**

The `drawPoint()` function draws a pixel at given coordinates (zero at top left).

The `lcd draw` functions can be called only from the functions with explicit LCD support (e.g. form and telemetry `printFunction`).

**Parameters**
- `<x>` (integer) – X-coordinate.
- `<y>` (integer) – Y-coordinate.

**Return value**
none

**Examples**

```lua
-- Prints the sinus function
local function printForm()
    lcd.setColor(200,200,200)
    lcd.drawLine(1,55,300,55)
    local r,g,b = lcd.getFgColor()
    lcd.setColor(r,g,b)
    for i = 1,300 do
        local val = math.sin(i*math.pi/150)
        lcd.drawPoint(i,val*50+55)
    end
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 13 - Points",nil, nil,printForm)
end

-------------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.0"}
```
**lcd.drawLine (<x1>, <y1>, <x2>, <y2>)**

The `drawLine()` function draws a line at given coordinates (zero at top left).

The `lcd draw` functions can be called only from the functions with explicit LCD support (e.g. form and telemetry `printFunction`).

**Parameters**

- **x1** (integer) – X-coordinate of the first point.
- **y1** (integer) – Y-coordinate of the first point.
- **x2** (integer) – X-coordinate of the second point.
- **y2** (integer) – Y-coordinate of the second point.

**Return value**

none

**Examples**

```lua
-- Prints a triangle
local function printForm()
    lcd.setColor(100,100,100)
    lcd.drawLine(10,10,300,50)
    lcd.drawLine(300,50,200,140)
    lcd.drawLine(10,10,200,140)
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 15 - Lines",nil, nil,printForm)
end

--------------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.0"}
```
**lcd.drawText (<x>, <y>, <text>[, <font = FONT_NORMAL>])**

The drawText () function draws a text at given top-left coordinates.

The lcd *draw* functions can be called only from the functions with explicit LCD support (e. g. form and telemetry printFunction).

**Parameters**

- **x** (integer) – X-coordinate.
- **y** (integer) – Y-coordinate.
- **text** (string) – specified text.
- **font** (integer) – one of the **Font definition** values: FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI

**Return value**

- none

**Examples**

```lua
-- Prints Several text messages
local text1="Short text"
local text2="Bold text, align right"
local text3="Large centered text"
local text4="Largest text"
local text5="Smallest text"

local function printForm()
    lcd.drawText(0,10,text1)
    -- Right aligned text
    lcd.drawText(310 - lcd.getTextWidth(FONT_BOLD,text2),10,text2,FONT_BOLD)
    -- Centered text
    lcd.drawText((310 - lcd.getTextWidth(FONT_BIG,text3))/2,40,text3,FONT_BIG)
    lcd.drawLine(0,85, 310,85)
    lcd.drawText(0,90,text4, FONT_MAXI)
    lcd.drawText(0,130,text5, FONT_MINI)
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 16 - Text",nil, nil,printForm)
end
--------------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.0"}
```
**lcd.drawRectangle** (<x>, <y>, <width>, <height>[, <radius = 0>])

The drawRectangle () function draws a rectangle, optionally with rounded corners.

The **lcd draw** functions can be called only from the functions with explicit LCD support (e.g. form and telemetry printFunction).

### Parameters
- x (integer) – X-coordinate.
- y (integer) – Y-coordinate.
- width (string) – rectangle width.
- height (integer) – rectangle height.
- radius (integer) – optional border radius.

### Return value
none

### Examples

```lua
-- Draw rectangles
local function printForm()
    lcd.drawRectangle(10,10,300,20)
    lcd.setColor(200,0,0)
    lcd.drawRectangle(10,40,300,20,5)
    lcd.setColor(0,200,0)
    lcd.drawRectangle(10,70,300,40,20)
end

local function init()
    system.registerForm(1,MENU_MAIN,"Test 16 - Text",nil, nil,printForm)
end
```

```
return {init=init, author="JETI model", version="1.0"}
```
**lcd.drawFilledRectangle (<x>,<y>,<width>,<height>[,<alpha=255>],[,<fontSpecs=0>])**

The drawFilledRectangle () function draws a rectangle filled with color, optionally with transparency.

The lcd draw functions can be called only from the functions with explicit LCD support (e. g. form and telemetry printFunction).

**Parameters**
- x (integer) – X-coordinate.
- y (integer) – Y-coordinate.
- width (integer) – rectangle width.
- height (integer) – rectangle height.
- alpha (integer) – transparency value (“0” means fully transparent).
- fontSpecs (integer) – added in DC/DS V4.22. Usefull only for B/W displays and has no impact on color displays (DC/DS-24). You can use one of these constants:
  - FONT_REVERSED – inverted text color,
  - FONT_GRAYED – bitmap is rasterized,
  - FONT_XOR – the bitmap is xored with background,
  - FONT_OR – the bitmap is ored with background,
  - FONT_AND.

**Return value**
none

**Examples**

```lua
-- Filled rectangles
local function printForm()
  lcd.drawFilledRectangle(10,10,300,20)
  lcd.setColor(200,0,0)
  lcd.drawFilledRectangle(10,40,300,20)
  lcd.setColor(0,200,0)
  lcd.drawFilledRectangle(10,70,300,40,50)
end

local function init()
  system.registerForm(1,MENU_MAIN,"Test 18 - Filled Rectangles",nil, nil,printForm)
end
```

```lua
return {init=init, author="JETI model", version="1.0"}
```
**lcd.drawCircle (<x>, <y>, <radius>)**

Since V4.20

The drawCircle () function draws a circle with defined radius.

The lcd draw functions can be called only from the functions with explicit LCD support (e. g. form and telemetry printFunction).

**Parameters**
- x (integer) – X-coordinate of the center.
- y (integer) – Y-coordinate of the center.
- radius (integer) – radius.

**Return value**
none

**Examples**
**lcd.drawEllipse** (<x>, <y>, <width>, <height>)

Since V4.20

The drawEllipse () function draws an ellipse with defined width and height.

The lcd draw functions can be called only from the functions with explicit LCD support (e.g. form and telemetry printFunction).

**Parameters**

- x (integer) – X-coordinate of the center.
- y (integer) – Y-coordinate of the center.
- width (integer) – ellipse width.
- height (integer) – ellipse height.

**Return value**

none

**Examples**
**lcd.drawNumber (<x>,<y>,<number> [,<font=FONT_NORMAL>])**

The drawNumber () function draws an integer number at specified coordinates.

The `lcd` draw functions can be called only from the functions with explicit LCD support (e.g. form and telemetry `printFunction`).

**Parameters**
- `x` (integer) – X-coordinate.
- `y` (integer) – Y-coordinate.
- `number` (integer) – number to be drawn.
- `font` (integer) – one of the Font definition values: FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI

**Return value**
none

**Examples**

```lua
-- Draws "120"
lcd.drawNumber (10,10,120)
lcd.drawNumber (10,30,120,FONT_BOLD)
```
**lcd.drawImage (<x>,<y>,<image> [,<alpha=255>])**

The drawImage () function draws an image that has already been loaded using loadImage() function.

The lcd draw functions can be called only from the functions with explicit LCD support (e.g. form and telemetry printFunction).

**Parameters**

- **x** (integer) – X-coordinate.
- **y** (integer) – Y-coordinate.
- **image** (table or string) – image to be drawn.
  - If the image is defined by a table, it must have been loaded by the lcd.loadImage() function. The table has the following structure:
    - **width** (integer) – horizontal portion of the image to be drawn.
    - **height** (integer) – vertical portion of the image to be drawn.
    - **data** (ImageData) – loaded image data.
  - If the image is defined by a string, an internal image will be used. See the
Default system images table.

- **alpha** (integer) – optional transparency value (“0” means fully transparent image).

**Return value**

none

Note: The function has no effect on DC/DS-16.

**Examples**

-- Drawing images with and without transparency
local imagepng, imagejpg

local function printForm()
  if(system.getTime()%2==0) then
    if(imagepng) then
      lcd.drawImage((310-imagepng.width)/2, 10, imagepng)
    end
  else
    if(imagejpg) then
      lcd.drawImage((310-imagejpg.width)/2, 10, imagejpg)
    end
  end
end

local function init()
  system.registerForm(1, MENU_MAIN, "Test 19 - Images", nil, nil, printForm)
  imagepng = lcd.loadImage("Apps/img/mx-2.png")
  imagejpg = lcd.loadImage("Apps/img/mx-2s.jpg")
end

return {init=init, author="JETI model", version="1.0"}

**lcd.loadImage (<absolutePath>)**

The loadImage () function loads an image from SD card to RAM.

Supported image formats:

- PNG (up to 320 × 240px), transparency supported.
- JPG (up to 1024 × 768px), “baseline” profile supported.

Please note that loading large images can take some time, which may result in worse user experience.

**Parameters**

- **absolutePath** (string) – full path to the file.

**Return value**

- **image** (table) – loaded image in case of success. The table has the following structure:
  - **width** (integer) – image width.
- **height** (integer) – image height.
- **data** (ImageData) – loaded binary image data.
- **nil** – in case of error.

Note: The function has no effect on DC/DS-16.

**Examples**

See `lcd.drawImage (<x>, <y>, <image> [, <alpha=255>])`
**lcd.getTextHeight (<font>)**

The `getTextHeight()` function gets the height of a given font.

**Parameters**
- **font** (integer) – one of the Font definition values: FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI

**Return value**
- (integer) – text height in pixels.

**Examples**
```lua
local height = lcd.getTextHeight(FONT_BOLD)
```
**lcd.getTextWidth (<font>, <text> [, maxCharacters])**

The getTextWidth () function returns the width of a specified text.

**Parameters**

- **font** (integer) – one of the **Font definition** values: FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI.
- **text** (string) – measured text.
- **maxCharacters** (integer) – if specified, tells the function to use up to maxCharacters of the specified text.

**Return value**

- (integer) – text width in pixels.

**Examples**

```lua
local width = lcd.getTextWidth(FONT_BOLD,"Some text")

-- Uses 4 characters ("Some") for size measurement
local width2 = lcd.getTextWidth(FONT_BOLD,"Some text",4)
```
**lcd.getBgColor ()**

The `getBgColor()` function returns the background color.

**Parameters**

none

**Return value**

- red, green, blue (integer) – background color.

**Examples**

```lua
local r, g, b = lcd.getBgColor()
```
**lcd.getFgColor ()**

The getFgColor () function returns the foreground color.

**Parameters**

none

**Return value**

- red, green, blue (integer) – foreground color.

**Examples**

```lua
local r, g, b = lcd.getFgColor()
```
**lcd.setClipping (<x>, <y>, <width>, <height>)**

Since V4.20

The setClipping() function sets a clipping rectangle so that drawing is possible only inside this rectangle.

The lcd draw functions can be called only from the functions with explicit LCD support (e.g. form and telemetry printFunction).

**Parameters**
- x (integer) – X-coordinate.
- y (integer) – Y-coordinate.
- width (string) – clipping rectangle width.
- height (integer) – clipping rectangle height.

**Return value**
none

**Examples**
**lcd.resetClipping ()**

Since V4.20

The resetClipping () function clears the clipping rectangle.

The lcd draw functions can be called only from the functions with explicit LCD support (e. g. form and telemetry printFunction).

**Parameters**

none

**Return value**

none

**Examples**
form.addRow (<componentsInRow>)

The addRow () function creates a new row in the layout of an interactive Lua form. The created row may be filled with up to componentsInRow form components (e. g. labels, select boxes etc.).

The interactive form has a strict layout because of the need for easy user interaction using a 3D rotary button. It consists of form controls stacked in a vertical layout, row by row. You can either push the controls directly to the form (typically bold labels that highlight a particular section), or to the preceding row.

The transmitter is able to handle up to 200 form components at a time, this limit prevents from enormous usage of system resources.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- componentsInRow (integer) – number of succeeding components that will be added into the row. Can be a value from 1 to 8.

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the row using this index.
- nil – in case of failure.

Examples

```lua
local appName = "Test 20 - Rows"
-- Form initialization
local function initForm()
  -- Create a new row in a layout. The row contains two components.
  form.addRow(2)
  -- Add components to the row.
  form.addLabel({label="Left align"})
  form.addLabel({label="Right align",alignRight=true})
  -- The next label will be added below the previous row.
  form.addLabel({label="Separate label, non-focusable"})
end

local function init()
  system.registerForm(1, MENU_MAIN, appName, initForm)
end
```

```
<table>
<thead>
<tr>
<th>Test 20 - Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left align</td>
</tr>
<tr>
<td>Separate label, non-focusable</td>
</tr>
</tbody>
</table>
```

```
return {init=init, author="JETI model", version="1.0", name=appName}
```
**form.addSpacer (<width>, <height>)**

The `form.addSpacer()` function creates a new spacer item in the layout of an interactive Lua form. The spacer reserves a blank space (width \times height pixels) in the form and acts like a visual separator.

The *form* library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the *form* library functions at any time.

**Parameters**

- **width** (integer) – item visual width.
- **height** (integer) – item visual height.

**Return value**

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

**Examples**

```lua
local appName = "Test 21 - Spacer"

-- Form initialization
local function initForm()
    -- Create a new row in a layout. The row contains 3 components.
    form.addRow(3)

    -- Add components to the row.
    form.addLabel({ label="Column 1", width=80 })
    form.addSpacer(150, 10)
    form.addLabel({ label="Column 2", width=80 })

    -- Add some space below
    form.addSpacer(300, 100)

    form.addRow(1)
    form.addLabel({ label="Note at the bottom of the screen", font=FONT_MINI, alignRight=true })
end

local function init()
    system.registerForm(1, MENU_MAIN, appName, initForm)
end

-----------------------------------------------------------------------------
return {init=init, author="JETI model", version="1.0", name=appName}
```
form.addLabel (<paramTable>)

The addLabel () function creates a new label in the layout of an interactive Lua form. The label usually statically describes another form component.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- paramTable (table) – label parameters table. All parameters are optional. It has the following structure:
  - label (string) – a textual description.
  - font (integer) – selected text font (FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI). [Default FONT_NORMAL]
  - alignRight (boolean) – true (align right) or false (align left). [Default false]
  - enabled (boolean) – true if the component is enabled and can be focused. [Default true]
  - visible (boolean) – component visibility. [Default true]
  - width (integer) – component width in pixels. [Default: automatically determined by the layout].

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples
See the form.addRow (<componentsInRow>)
form.addLink ([clickedCallback], paramTable])

The addLink () function creates a new visual link in the layout of an interactive Lua form. The link can be used to trigger some actions when the user clicks it using 3D rotary button. Using links the user is able to easily navigate through the application subforms.

You can use double arrows to make the navigation easily visible and usable:

“Link to second form >>”
“<< Link back”

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- clickedCallback (function) - optional function that is called after the user clicks the link. The function call contains zero parameters.
- paramTable (table) – label parameters table. All parameters are optional. It has the following structure:
  - label (string) – a textual link description.
  - font (integer) – selected text font (FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI). [Default FONT_NORMAL]
  - alignRight (boolean) – true (align right) or false (align left). [Default false]
  - enabled (boolean) – true if the component is enabled an can be focused. [Default true]
  - visible (boolean) – component visibility. [Default true]
  - width (integer) – component width in pixels. [Default: automatically determined by the layout].

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples

See the system.registerForm ( <formNo>,<parentMenuID>, <label>, <initFunction>, <keyPressFunction>, <printFunction> )
form.addIntbox (<value>, <minimum>, <maximum>, <defaultValue>, <decimals>, <step>[, <changedCallback>, <paramTable>])

The addIntbox () function creates a new integer editor in the layout of an interactive Lua form. The intbox is able to represent integer values from -32768 to 32767. You can specify the number of decimals as well but, however, the intbox API works only with integer values. For example if you want to set the intbox value to “5.0”, you should specify “50” as the current value and “1” as number of decimals. If the user changes the intbox value, it will not return “5.1” but “51” instead.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- **value** (integer) – current value. It must be within range (-32768, 32767).
- **minimum, maximum** (integer) – hardcoded numerical limits, must be within range (-32768, 32767).
- **defaultValue** (integer) – default value is set after the 3D rotary button has been pressed for a long time. Must be within range (-32768, 32767).
- **decimals** (integer) – number of digits to be displayed behind the decimal point (0-6).
- **step** (integer) – a single-step increment (1 by default).
- **changedCallback** (function) – optional function that is called after the user modifies the intbox value. The function call contains a single integer parameter – new value.
- **paramTable** (table) – parameters table. All parameters are optional. It has the following structure:
  - **font** (integer) – selected text font (FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI). [Default FONT_NORMAL]
  - **enabled** (boolean) – true if the component is enabled an can be focused. [Default true]
  - **visible** (boolean) – component visibility. [Default true]
  - **width** (integer) – component width in pixels. [Default: automatically determined by the layout].
  - **label** (string) – a label appended to the number, usually a telemetry unit (since V4.22).

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples

See the `system.pLoad` ( <param> [, <defaultValue>] )
The `form.addSelectbox` function creates a new selection editor (_selectbox_) in the layout of an interactive Lua form. The selectbox allows you to pick one item from a list of the specified options.

The `form` library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the `form` library functions at any time.

**Parameters**

- **values** (table) – a table with specified options. All indexes within this table must be numeric, contiguous, and the table must begin with index 1.
- **currentIndex** (integer) – index of currently selected option.
- **enableForm** (boolean) – if true, the options will be offered in a standalone dialogue.
- **changedCallback** (function) – optional function that is called automatically after the user modifies the selectbox value. The function call contains a single integer parameter – a new index.
- **paramTable** (table) – parameters table. All parameters are optional. It has the following structure:
  - **font** (integer) – selected text font (FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI). [Default FONT_NORMAL]
  - **enabled** (boolean) – true if the component is enabled and can be focused. [Default true]
  - **visible** (boolean) – component visibility. [Default true]
  - **width** (integer) – component width in pixels. [Default: automatically determined by the layout].

**Return value**

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

**Examples**

See `form.addAudioFilebox` (<currentAudioFile>[, <changedCallback>, <paramTable>])
form.addAudioFilebox (<currentAudioFile>, <changedCallback>, <paramTable>)

The addAudioFilebox () function creates a new drop-down menu of available audio files (audiofilebox). The audiofilebox is then pushed into the layout of an interactive Lua form. The audiofilebox allows you to pick a single audio file from a list of all available audio files. It searches for *.WAV file in the following directories:

- /Audio,
- /Audio/XX, where XX stands for a current language code.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- **currentAudioFile** (string) – currently selected audio file. Empty string if not specified.
- **changedCallback** (function) – optional function that is called automatically after the user modifies the audiofilebox value. The function call contains a single string parameter – a new selected audio file.
- **paramTable** (table) – parameters table. All parameters are optional. It has the following structure:
  - **font** (integer) – selected text font (FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI). [Default FONT_NORMAL]
  - **enabled** (boolean) – true if the component is enabled and can be focused. [Default true]
  - **visible** (boolean) – component visibility. [Default true]
  - **width** (integer) – component width in pixels. [Default: automatically determined by the layout].

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples

```lua
local appName="Test 22 - Audio play"
local playedFile
local playedType = 1
local switch
local prevVal = 1

local typeOptions={"Play in background", "Play immediately", "Add to queue"}
local typeValues={AUDIO_BACKGROUND, AUDIO_IMMEDIATE, AUDIO_QUEUE}

--------------------------------------------------------------------
local function fileChanged(value)
    playedFile=value
    system.pSave("file",value)
end
```

local function typeChanged(value)
    playedType=value
    system.pSave("type",value)
end

local function switchChanged(value)
    switch=value
    system.pSave("switch",value)
end

local function initForm(formID)
    form.addRow(2)
    form.addLabel({label="Select file"})
    form.addAudioFilebox(playedFile or ",", fileChanged)

    form.addRow(2)
    form.addLabel({label="Playback type",width=120})
    form.addSelectbox(typeOptions,playedType or 1,false,typeChanged,{width=190})

    form.addRow(2)
    form.addLabel({label="Switch"})
    form.addInputbox(switch,true,switchChanged)
    form.setButton(1,"Play",ENABLED)
end

local function keyPressed(key)
    if(key==KEY_1) then
        system.playFile(playedFile,typeValues[playedType])
    end
end

-- Init function
local function init()
    system.registerForm(1,MENU_MAIN,appName,initForm,keyPressed);
    playedFile = system.pLoad("file","")
    switch = system.pLoad("switch")
    playedType = system.pLoad("type",1)
end

-- Loop function
local function loop()
    local val = system.getInputsVal(switch)
    if(val and val>0 and prevVal==0) then
        system.playFile(playedFile,typeValues[playedType])
    prevVal=1
    elseif(val and val<=0) then
        prevVal=0
    end
end

return { init=init, loop=loop, author="JETI model", version="1.00",name=appName}
form.addTextbox (<currentText>, <maxCharacters> [, <changedCallback>, <paramTable>])

The addTextbox () function creates a new text editor (textbox) in the layout of an interactive Lua form. The textbox allows you to edit short texts (e.g. names and labels).

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters
- **currentText** (string) – a text currently displayed.
- **maxCharacters** (integer) – maximum number of characters that are allowed (1-63). Please note that some special UTF-8 characters take place of two standard ASCII characters.
- **changedCallback** (function) – optional function that is called automatically after the user modifies the textbox value. The function call contains a single string parameter – a new text.
- **paramTable** (table) – parameters table. All parameters are optional. It has the following structure:
  - **font** (integer) – selected text font (FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI). [Default FONT_NORMAL]
  - **enabled** (boolean) – true if the component is enabled and can be focused. [Default true]
  - **visible** (boolean) – component visibility. [Default true]
  - **width** (integer) – component width in pixels. [Default: automatically determined by the layout].

Return value
- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples
See the system.pLoad ( <param> [, <defaultValue>] )
form.addInputbox (<selectedSwitch>, <enableProportional> [, <changedCallback>, <paramTable >])

The addInputbox () function creates a new input selection box (inputbox) in the layout of an interactive Lua form. The inputbox allows user to assign an arbitrary physical control (stick, switch, and knob) or virtual control (logical switch, sequencer...) so that the Lua application is able to read its value using system.getInputsVal (<Input 1>[, <Input 2>] [, <Input 3>], ...) function.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- **selectedSwitch** (SwitchItem) – currently assigned control item (or nil if it has not been assigned).
- **enableProportional** (boolean) – if true, the user can specify both binary switches (“on/off”) and proportional controls as the input. Otherwise, only binary switches (“on/off”) are allowed.
- **changedCallback** (function) – optional function that is called automatically after the user modifies the selected switch. The function call contains a single SwitchItem parameter – a new assigned switch.
- **paramTable** (table) – optional parameters table. All parameters are optional. It has the following structure:
  - **font** (integer) – selected text font (FONT_NORMAL, FONT_BOLD, FONT_MINI, FONT_BIG, FONT_MAXI). [Default FONT_NORMAL]
  - **enabled** (boolean) – true if the component is enabled an can be focused. [Default true]
  - **visible** (boolean) – component visibility. [Default true]
  - **width** (integer) – component width in pixels. [Default: automatically determined by the layout].

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples

See the system.pLoad ( <param> [, <defaultValue>] )

Version 1.2, all rights reserved. February 16, 2017 [89]
form.addCheckbox (<checked> [, <clickedCallback>, <paramTable>])

The addCheckbox () function creates a new visual checkbox in the layout of an interactive Lua form. The checkbox can represent a single Boolean value using a tick mark or cross mark.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- **checked** (boolean) – true if the checkbox is checked.
- **clickedCallback** (function) - optional function that is called after the user clicks the checkbox. The function call contains a Boolean value of the checkbox state.
- **paramTable** (table) – label parameters table. All parameters are optional. It has the following structure:
  - **enabled** (boolean) – true if the component is enabled an can be focused. [Default true]
  - **visible** (boolean) – component visibility. [Default true]
  - **width** (integer) – component width in pixels. [Default: automatically determined by the layout].

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples
local appName="Test 28 - Checkbox/Servo Test"
local showServo=true
local componentIndex

---
local function checkClicked(value)
    showServo = not value
    form.setValue(componentIndex,showServo)
end
---
local function initForm(formID)
    form.addRow(2)
    form.addLabel({label="Show servo outputs",width=270})
    componentIndex = form.addCheckbox(showServo,checkClicked)
end
---
local function printForm()
    if(not showServo) then
        return
    end
    local s1,s2,s3,s4,s5,s6,s7,s8 = system.getInputs("O1","O2","O3",
        "O4","O5","O6","O7","O8")
    local values={s1,s2,s3,s4,s5,s6,s7,s8}
    local offset=25
    local offsetx=10
    local textVal
    for i=1,8 do
        lcd.drawText(offsetx,offset,string.format("Ch %d:",i))
        textVal = string.format("%.1f %%",values[i]*100)
        lcd.drawText(offsetx+130-lcd.getTextWidth(FONT_NORMAL,textVal),
            offset,textVal)
        offset=offset + 20
        if(i==4) then
            offsetx = offsetx + 155
            offset = 25
        end
    end
end
---
-- Init function
local function init()
    system.registerForm(1,MENU_MAIN,appName,initForm,nil ,printForm);
end
---
return { init=init, author="JETI model", version="1.00",name=appName}
form.getValue (<componentIndex>)

The `getValue()` function returns the value of a form component at a given index. You can retrieve the value of an intbox, selectbox, audiofilebox, inputbox and textbox.

The `form` library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the `form` library functions at any time.

**Parameters**
- `componentIndex` (integer) – an index of the component that was returned by calling the appropriate `form.addXXX()` function.

**Return value**
- (integer) – for intbox returns the current numeric value.
- (integer) – for selectbox returns the current index.
- (string) – for textbox returns the current text.
- (string) – for audiofilebox returns the current filename.
- (`SwitchItem`) – for inputbox returns the current switch.
- `nil` – in case of failure (component doesn’t exist or it is not of the supported types).

**Examples**

```lua
local appName="Test 23 - Buttons"
local intIdx,timeIdx

--------------------------------------------------------------------
local function valueChanged(val)
    if(val==0) then
        form.setButton(1,"<<",DISABLED)
        form.setButton(2,">>",ENABLED)
    elseif(val==100) then
        form.setButton(1,"<<",ENABLED)
        form.setButton(2,">>",DISABLED)
    else
        form.setButton(1,"<<",ENABLED)
        form.setButton(2,">>",ENABLED)
    end
end

--------------------------------------------------------------------
local function initForm(formID)
    form.addRow(2)
    form.addLabel({label="Select value"})
    intIdx = form.addIntbox(0,0,100,0,0,1,valueChanged)
    form.addRow(2)
    form.addLabel({label="Timestamp"})
```

timeIdx = form.addIntbox(0,0,32000,0,0,1,nil,{enabled=false})

form.setButton(1,"<<",DISABLED)
form.setButton(2,">>",ENABLED)
end

--------------------------------------------------------------------
local function keyPressed(key)
    local val = form.getValue(intIdx)
    if(key==KEY_1) then
        if(val>0) then
            val= val-1
            form.setValue(intIdx,val)
        end
    elseif(key == KEY_2) then
        if(val<100) then
            val= val+1
            form.setValue(intIdx,val)
        end
    end
end

local function printForm(key)
    local value = form.getValue(intIdx)
    lcd.drawText(10,50,value.."%",FONT_MAXI)
end

--------------------------------------------------------------------
-- Init function
local function init()
    system.registerForm(1,MENU_MAIN,appName,initForm,keyPressed,printForm);
end

local function loop()
    if(timeIdx) then
        form.setValue(timeIdx,system.getTimeCounter()//1000)
    end
end

--------------------------------------------------------------------
return { init=init, loop=loop, author="JETI model", version="1.00",name=appName}
**form.setValue (<componentIndex>,<newValue>)**

The `setValue()` function sets the value of a form component at a given index. You can set a new value to an intbox, selectbox, audiofilebox, inputbox and textbox.

Please note that if the form component doesn’t exist or if the data type of a new value is different from what is needed, an error is generated.

The `form` library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the `form` library functions at any time.

**Parameters**

- **componentIndex** (integer) – an index of the component that was returned by calling the appropriate `form.addXXX()` function.
- **newValue** – one of the required data types:
  - (integer) – for intbox, new value.
  - (integer) – for selectbox, new index.
  - (string) – for textbox, new text.
  - (string) – for audiofilebox, new filename (must exist).
  - (SwitchItem) – for inputbox, new switch (or nil if you want to clear the assigned switch).

**Return value**

none

**Examples**

See `form.setValue()`
**form.setProperties (<componentIndex>, <paramTable>)**

The `setProperties()` function sets the properties of a form component at a given index. You can set new properties to all form controls.

Please note that if the form component doesn’t exist, an error is generated.

The `form` library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the `form` library functions at any time.

**Parameters**
- `componentIndex` (integer) – an index of the component that was returned by calling the appropriate `form.addXXX()` function.
- `paramTable` – a table with mixture of supported parameters:
  - `visible` (boolean) – specifies the form item visibility.
  - `enabled` (boolean) – specifies that the form item can be focused.
  - `label` (string) – specifies a new text label (supported only by `label`, `link` and `intbox`).
  - `alignRight` (boolean) – true for right-aligned text (supported only by `label` and `link`).

**Return value**

none

**Examples**

See `form.close()`
**form.setButton (<buttonNo>, <text>[,<newState = DISABLED>])**

The setButton () function sets the properties of a specified function button F(1) – F(5). You can set short text labels and button states.

The *form* library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the *form* library functions at any time.

**Parameters**

- **buttonNo** (integer) – button identifier (1-5).
- **text** (string) – button text, maximum 7 characters are allowed to be displayed. If the text begins with a colon character, e.g. “:file” it will be replaced by an internal image, see the
- **Default** system images table.
- **newState** (integer) – new button state, one of the following options:
  - **DISABLED** (0) – the button is disabled (but visible) and should be considered inactive.
  - **ENABLED** (1) – the button is enabled and active.
  - **HIGHLIGHTED** (2) – the button is enabled and activated/highlighted.
  - (3-255) – the button is disabled and hidden.

**Return value**
none

**Examples**
See `form.setButton <buttonNo>, <text>[,[newState = DISABLED]>`)
**form.getButton (<buttonNo>)**

The `getButton()` function retrieves the properties of a specified function button F(1) – F(5). The button text and state are returned in order.

The *form* library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the *form* library functions at any time.

**Parameters**
- **buttonNo** (integer) – button identifier (1-5).

**Return value**
- **text** (string), **state** (integer) – button text and state in a row. The state can be one of the following options:
  - **DISABLED** (0) – the button is disabled (but visible) and should be considered inactive.
  - **ENABLED** (1) – the button is enabled and active.
  - **HIGHLIGHTED** (2) – the button is enabled and activated/highlighted.
  - **(3)** – the button is disabled and hidden.

**Examples**

```lua
local appName="Test 24 - Mode"
local mode=1

local function checkButtons()
    form.setButton(1,"1",mode==1 and HIGHLIGHTED or ENABLED)
    form.setButton(2,"2",mode==2 and HIGHLIGHTED or ENABLED)
    form.setButton(3,"3",mode==3 and HIGHLIGHTED or ENABLED)
    form.setButton(4,"4",mode==4 and HIGHLIGHTED or ENABLED)
end

local function initForm(formID)
    form.setButton(5,"Test",ENABLED)
    checkButtons()
end

local function printForm()
    lcd.drawText(10,50,"Tx Mode: ".mode,FONT_MAXI)
end
```

Version 1.2, all rights reserved. February 16, 2017
local function keyPressed(key)
    if key == KEY_1 then
        mode = 1
    elseif key == KEY_2 then
        mode = 2
    elseif key == KEY_3 then
        mode = 3
    elseif key == KEY_4 then
        mode = 4
    elseif key == KEY_5 then
        form:preventDefault()
        local text, state = form:getButton(5)
        form:setButton(5, text, (state == HIGHLIGHTED) and ENABLED or HIGHLIGHTED)
    end
    checkButtons()
end

-- Init function
local function init()
    system:registerForm(1, MENU_MAIN, appName, initForm, keyPressed, printForm);
end

return { init = init, author = "JETI model", version = "1.00", name = appName }
form.getActiveForm ()

The getActiveForm () function retrieves the ID of an active form. If there is no application form running, returns nil.

Parameters
none

Return value
- (integer) – ID of an active form (0, 1 or 2)
- nil – if the application has no active form

Examples
See form.close ()
form.close()

The close() function closes the interactive Lua form and clears the resources. The form will not be closed immediately after calling this function, however, it will be closed during the nearest opportunity.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters
none

Return value
none

Examples

```lua
local appName="Test 25 - Open/Close form"
local lastTimeChecked
local rowIndex

-- Function to initialize the form
local function initForm(formID)
    rowIndex = form.addRow(1)
    form.addLabel({label="Blinking label"})
end

-- Function to react to key presses
local function keyPressed(key)
    if(key~=KEY_RELEASE) then
        form.close()
    end
end

-- Loop function
local function loop()
    if( form.getActiveForm() ) then
        lastTimeChecked = system.getTimeCounter()
        form.setProperties(rowIndex,{visible = (lastTimeChecked//1000)%2==0})
    else
        if( system.getTimeCounter() > lastTimeChecked + 2000) then
            -- Show the form immediately
            system.registerForm(1,0,appName,initForm,keyPressed);
        end
    end
end

-- Init function
local function init()
    lastTimeChecked = system.getTimeCounter()
end

return { init=init, loop=loop, author="JETI model", version="1.00", name=appName}
```
form.reinit ([<subFormID = 1>])

The reinit () function forces the interactive Lua form to reinitialize. All resources (form components and buttons) are cleared before reinitialization. The registered initForm(subFormID) function is called in the reinitialization process. The form will not be reinitialized immediately after calling this function but as soon as Lua finishes execution.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters
- subFormID (integer) – identifier of the subform (1-128). Using this identifier you can create several subforms within the same application.

Return value
- none

Examples
See the examples in system.registerForm ( <formNo>, <parentMenuID>, <label>, <initFunction>, <keyCodeFunction>, <printFunction> ) function description.
The `form.preventDefault()` function prevents the default behavior after the user presses one of the specific buttons:

<table>
<thead>
<tr>
<th>Button</th>
<th>Default behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_MENU</td>
<td>Debug console is displayed over the active form.</td>
</tr>
<tr>
<td>KEY_5</td>
<td>The form is closed.</td>
</tr>
<tr>
<td>KEY_ESC</td>
<td>The form is closed.</td>
</tr>
<tr>
<td>KEY_POWER</td>
<td>The form is closed.</td>
</tr>
</tbody>
</table>

The `form` library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the `form` library functions at any time.

**Parameters**
- none

**Return value**
- none

**Examples**
See `form.getButton(<buttonNo>)`
The `form.waitForRelease()` function prevents any succeeding `keyPress` events until all buttons are released.

The `form` library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the `form` library functions at any time.

**Parameters**
none

**Return value**
none

**Examples**
**form.setFocusedRow (<rowNumber>)**

The setFocusedRow () function sets a focus to the selected row number.

The *form* library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the *form* library functions at any time.

**Parameters**
- (integer) – row to be focused (1 – N).

**Return value**
- none

**Examples**
**form.getFocusedRow ()**

The getFocusedRow () function returns currently highlighted row in an interactive form.

The *form* library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the *form* library functions at any time.

**Parameters**

none

**Return value**

- (integer) – focused row (1 – N if the form contains any components, 0 otherwise).

**Examples**
form.addIcon (<path> [, <paramTable>])

Since V4.20

The addIcon () function creates a new visual icon in the layout of an interactive Lua form. The icon is specified by its path in the filesystem. JPG and PNG files are supported in DC/DS-24. To preserve memory, the image is loaded and unloaded on demand, as soon as the component’s visibility changes.

The form library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the form library functions at any time.

Parameters

- **path** (string) – absolute path to the image file.
- **paramTable** (table) – label parameters table. All parameters are optional. It has the following structure:
  - **label** (string) – a label that is displayed below the icon.
  - **font** (font enum) – font used for label.
  - **enabled** (boolean) – true if the component is enabled and can be focused. [Default true]
  - **visible** (boolean) – component visibility. [Default true]
  - **width** (integer) – component width in pixels. [Default: automatically determined by the layout].
  - **height** (integer)
  - **padding** (integer)

Return value

- (integer) – component index within the form, in case of success. Later on you can reference the component using this index.
- nil – in case of failure.

Examples
**form.setTitle (<title>)**

Since V4.20

The setTitle () function sets or clears the title of the interactive Lua form.

The *form* library functions are enabled only if the appropriate Lua application interactive form is displayed. You cannot interfere with another Lua application form by calling the *form* library functions at any time.

**Parameters**

- *title* (string) – new form title. Up to 64 characters are supported.

**Return value**

none

**Examples**

```lua
-- Sets the new title
form.setTitle("New Form Title")

-- Clears the form title
form.setTitle("")
```
form.question (<boldText>,<textLine1 = "">, <textLine2 = "">,<timeoutms = 0>, <onlyInfo=false>,<timeoutBeforeOk = 0>)
Since V4.20

The question () function raises a question with a given highlighted text and additional two lines of description. The Lua caller function waits in blocking mode for the results of the question form.

The question function can be called even if the application form is not created.

Parameters
- **boldText** (string) – form question or informative text.
- **textLine1, textLine2** (string) – additional informative text strings.
- **timeoutms** (integer) – timeout in milliseconds, after which the question form automatically disappears. If the timeout is zero, it will not disappear.
- **onlyInfo** (boolean) – decides whether “YES”/”NO” buttons are available (false) or only “OK” button is present(true).
- **timeoutBeforeOk** (integer) – number of milliseconds that must elapse before the “Yes”/”No” buttons are enabled.

Return value
- 0 – “No” has been selected or a timeout has elapsed.
- 1 – “Yes” has been selected.
- -1 – an error has occurred (the question form could not be created).

Examples
-- Creates a form/question with 10s timeout, Yes/No buttons
-- and 2 seconds of “ready-time”
local res = form.question("Enable function?", "Function will be enabled", "Description of the function",10000,false,2000)
## Versions History

<table>
<thead>
<tr>
<th>Version</th>
<th>Tx firmware</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>4.10</td>
<td>07/2016</td>
<td>First release.</td>
</tr>
<tr>
<td>1.1</td>
<td>4.20</td>
<td>12/2016</td>
<td>Added functions: <code>lcd.setClipping</code>, <code>lcd.resetClipping</code>, <code>lcd.drawCircle</code>, <code>lcd.drawEllipse</code>, <code>form.addIcon</code>, <code>form.setTitle</code>, <code>form.question</code>, <code>system.getProperty</code>. Modified: <code>system.setProperty</code>. The <code>init()</code> function now has a single parameter, representing state of the transmitter. Limited Lua functionality in DC/DS-16. The drawing functions are limited to use only a single color, external images are forbidden. The DC/DS-16 can manage up to 2 applications and up to 4 Lua controls are available.</td>
</tr>
<tr>
<td>1.2</td>
<td>4.22</td>
<td>2/2017</td>
<td>Function <code>lcd.drawFilledRectangle</code> has now an additional optional parameter that defines rendering style on DC/DS-14/16. Optimizations for DC/DS-16/14: several of the libraries now are stored in read-only tables. Added functions: <code>system.getSwitchInfo</code>, <code>io.readall</code>. Added constants: <code>FONT_XOR</code>, <code>FONT_OR</code>, <code>FONT_GRAYED</code>, <code>FONT_AND</code>. Function <code>system.getDateTime</code> now returns a “dst” flag. Added “label” property for <code>Intbox</code>.</td>
</tr>
</tbody>
</table>
Lua Copyright Notice


Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Lua CJSON - JSON support for Lua

Copyright (c) 2010-2012  Mark Pulford <mark@kyne.com.au>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.