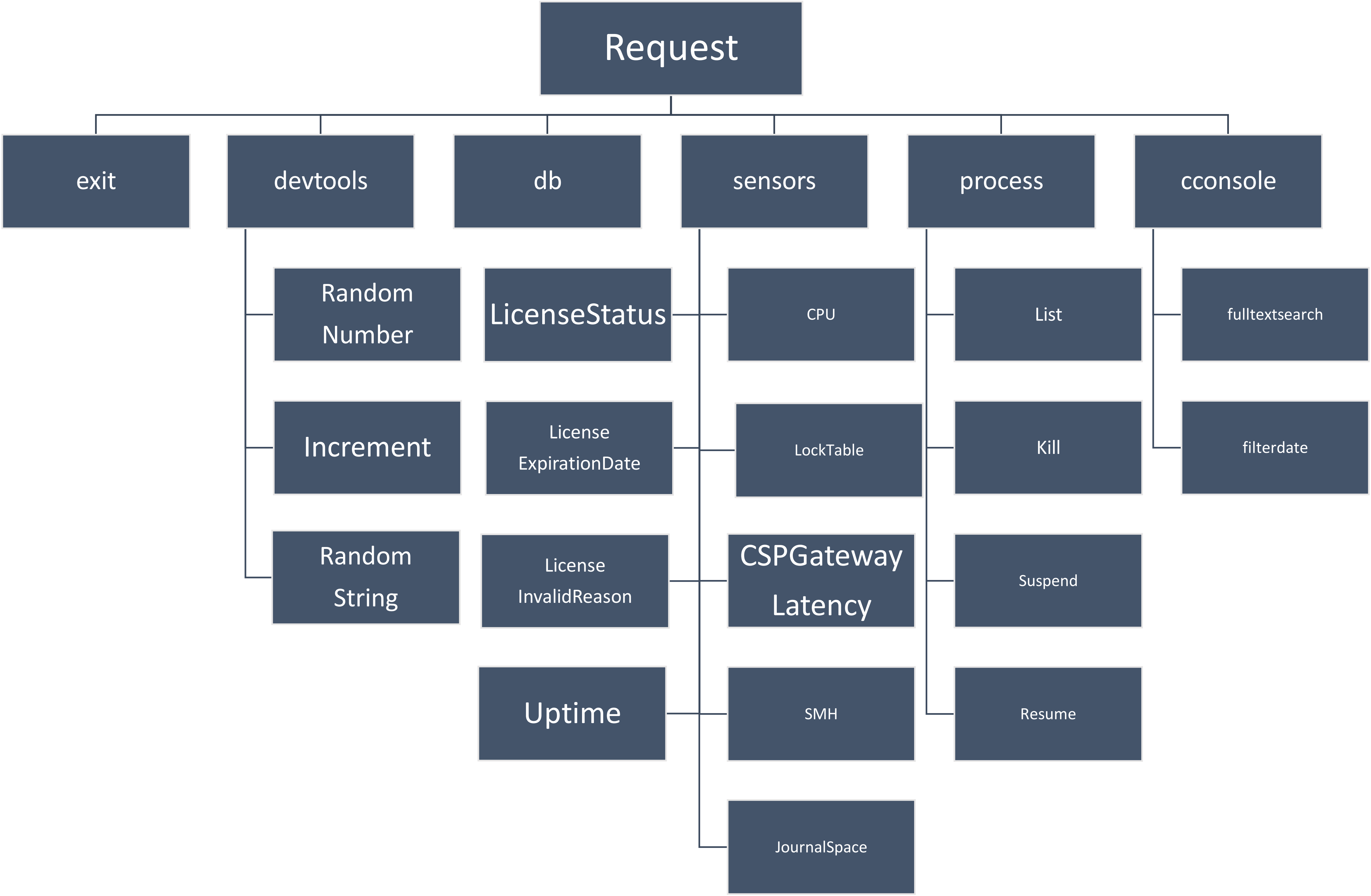

Request
exit
devtools
db
sensors
process
cconsole
Random Number
Increment
Random String
LicenseStatus
License ExpirationDate
License InvalidReason
Uptime
CPU
LockTable
CSPGateway Latency
SMH
JournalSpace
List
Kill
Suspend
Resume
fulltextsearch
filterdate

Caché Mobile Administrator Request map

Request	Description	Response Example
exit	Closes websocket	

db	Sends info about databases	{
		"Databases":[{
		"Path":"\/InterSystems\/Cache2014\/mgr\/cAdmin-Server\/",
		"DirectorySpace":22741,
		"DBWrites":0,
		"DBReads":0,
		"DBLatency":0,
		"DBStats":"130,3144"
		}, {
		"Path":"\/InterSystems\/Cache2014\/mgr\/cacheaudit\/",
		"DirectorySpace":22741,
		"DBWrites":0,
		"DBReads":0,
		"DBLatency":0,
		"DBStats":"11,7"
		}]
		}

Devtools		
devtools:RandomNumber	Returns random number	{"RandomNumber":22}
devtools:Increment	Increments number from 0 to 100, then starts from 0	{" Increment ":45}
devtools:RandomString	Returns random string	{"RandomString":"SJFNJSNJNDVJDNJNV"}
Sensors		
sensors:CPU,LockTable,CSPGatewayLatency,SMH,JournalSpace,Uptime,LicenseStatus,LicenseExpirationDate,LicenseInvalidReason	Various sensors data. In your query, list only the ones you need. About metrics see	{ "CPU":36, "LockTable":"0.34", "CSPGatewayLatency":5.457, "SMH":6, "JournalSpace":21204.34, "Uptime":12236.915142, "LicenseStatus":"Правильный", "LicenseExpirationDate":"04\/30\/2014",

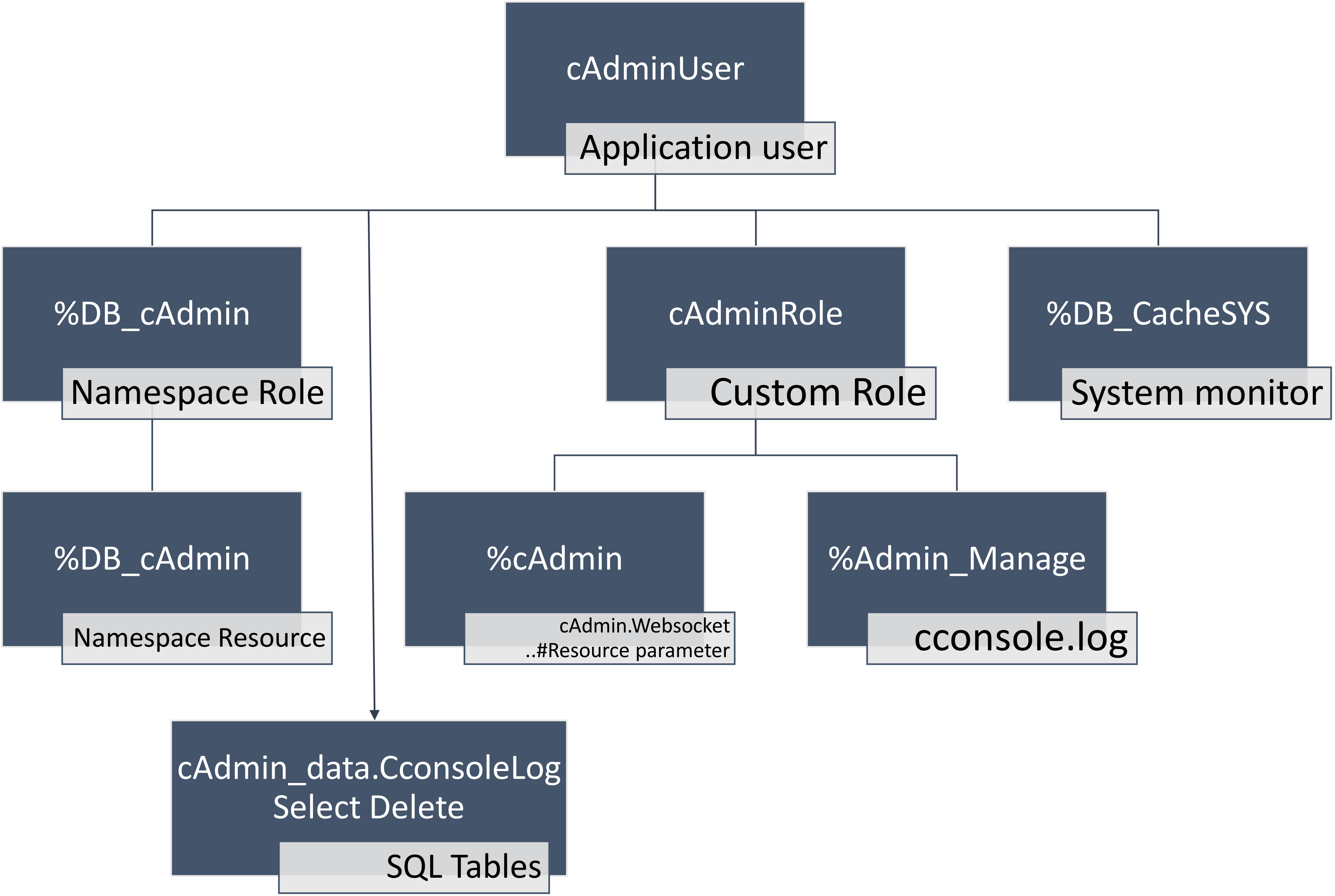
		"LicenseInvalidReason":"" }
Process		

process:List		{
		"processes": [{
		"id": "20037",
		"routine": "CONTROL"
		},
		{
		"id": "15568",
		"routine": "WRTDMN"
		}]
		}

process: Kill,<id>	Kill process with id=<id>	
process: Suspend,<id>	Suspend process with id=<id>	
process: Resume,<id>	Resume process with id=<id>	
Cconsole		

cconsole-fulltextsearch:‘Term1’ ,’Term2’	Search cconsolelog for all entries containing Term1 and Term2. Any number of search terms is possible	{
		"children": [{
		"ID": 127,
		"DateTime": "2013-09-02 14:09:37.22",
		"Message": "[SYSTEM MONITOR] CSPGatewayLatency(127.0.0.1:80) Alert: CSPGatewayLatency = 10002.574, 10002.574, 10002.39 (Max value is 2000).",
		"Pid": 6593,
		"Severity": 2
		},
		{
		"ID": 130,
		"DateTime": "2013-09-03 04:00:34.357",
		"Message": "[SYSTEM MONITOR] CSPGatewayLatency(127.0.0.1:80) Alert: CSPGatewayLatency = 10002.579, 10002.678, 10002.494 (Max value is 2000).",
		"Pid": 6593,
		"Severity": 2
		}]
		}

cconsole-filterdate:yyyy-mm-dd hh:mm:ss.fff, YYYY-MM-DD HH:MM:SS.FFF	Select all cconsole.log entries from yyyy-mm-dd hh:mm:ss.fff to YYYY-MM-DD HH:MM:SS.FFF	
cconsole-alert	Receive all new cconsole.log entries every 10 seconds	



cAdminUser

Application user

%DB_cAdmin

Namespace Role

cAdminRole

Custom Role

%DB_CacheSYS

System monitor

%DB_cAdmin

Namespace Resource

%cAdmin

cAdmin.Websocket
..#Resource parameter

%Admin_Manage

cconsole.log

cAdmin_data.CconsoleLog
Select Delete

SQL Tables

Caché Mobile Administrator Installation guide

1. Create new Namespace cAdmin with new database cAdmin with new access resource %DB_cAdmin
2. Into your new namespace cAdmin mirror package SYS from database CACHESYS
3. Create new resource %cAdmin with public USE on
4. Create new role cAdminRole, containing %cAdmin resource (step 3) and %Admin_Manage resource
5. Import all Cache classes from this repository https://github.com/eduard93/cAdmin-Server/ Either download zip or clone git
6. In cAdmin.WebSocket class modify resource parameter to match %cAdmin resource
7. Compile all classes
8. Create new user cAdminUser with roles %DB_cAdmin (step 1), cAdminRole (step 4) and %Admin_Manage role
9. For user cAdminUser on tab SQL Tables add access to Select and Delete for table cAdmin_data.CconsoleLog from cAdmin namespace

Use (JavaScript).

```
ws = new WebSocket("ws://<server ip>:<server port>/csp/<Namespace>
/cAdmin.WebSocket.cls");

ws.onopen = function(){ ws.send(JSON.stringify({User:" cAdminUser ",Password:"
cAdminUserPassword"}))};

ws.onmessage = function(m){console.log(m)};

ws.send(‘devtools:RandomNumber’);
```

In this example we, first, create new websocket object and point it to our websocket.

Next, we specify what to send to server after establishing the connection – JSON, containing authorization information.

After this use ws.send to send requests from request map

Use ws.readyState to check that WebSocket is working