commsignia

muci

The ultimate command line tool to manage Commsignia devices

Quick Start Guide

and

User Guide

15th January 2023

muci version 2.1.x

Contents

Introducing **muci**

Commsignia y20 introduced json configuration files to store settings previously managed by the uci toolset, and consequently the uci command can no longer be used to query / change these settings. Y20 is intended to be "GUI first", but command line administration is also important. Experience shows that users cannot be expected to edit json files directly. This led to the inception of **muci** – a command line tool that lets users edit json files without facing the complexities of editing json files.

Quick Start Guide

Before you start

This guide applies to **muci** version 2.0.x. Please check your version by typing **muci version**.

muci will make configuration changes immediately. However, unplugged-rt-restart.sh is still needed for the changes to take effect.

Example: how to set the station type

Let's say you want to change the station type. You know that this setting is stored in the "its" configuration, but you cannot remember the exact name of the property. Here's how you can find it:

muci its find <filters>

```
root@ITS-RS4-M-1001848:~# muci its find sta rol
stationInfo.role = "Unknown"
```

muci uses keys with a dotted notation to represent json nodes (see details later).

Now that we know that the key is `stationInfo.role`, we can change the value using:

muci its set <key> <value>

What if we do not know the possible values? – let's use **muci its set <key>** without specifying the <value>!

```
root@ITS-RS4-M-1001848:~# muci its set stationInfo.role
stationInfo.role = "Unknown"
Type: string
Select the role of the station.
Valid values for stationInfo.role:
Unknown
BasicVehicle
PublicTransport
SpecialTransport
...
```

Let's set the value:

```
root@ITS-RS4-M-1001848:~# muci its set stationInfo.role PublicTransport
root@ITS-RS4-M-1001848:~#
```

There are two ways to query the current value:

muci its find <filters> or **muci its get <key>**

muci find accepts multiple free-text filter keywords and matches case-insensitive on both keys and values. (See example above.)

muci get expects a single dotted key with proper capitalization (case-sensitive)

```
root@ITS-RS4-M-1001848:~# muci its get stationInfo.role
stationInfo.role = "PublicTransport"
```

Finding a key or value globally

When you do not know which config file contains the desired setting, you can run a global search for multiple keywords using **muci find:**

```
root@ITS-RS4-M-1001848:~# muci find port capt
its.capture.port = 7943
```

Note the "its." at the beginning of the line: it means that the key was found in its.json. This is called a ***topic***. (A topic is usually a combination of a json config file and the corresponding json schema, **muci** knows the filenames and locations so the user does not have to.) For a list of topics, just type **muci**:

```
root@ITS-RS4-M-1001848:~# muci
Usage: muci <topic>
Select a topic:
its: Generic ITS Settings
saf: Safety Application Settings
scms: SCMS Settings
cff: CFF Settings
tlc: Traffic Light Controller Settings
srm: SRM Processor (TSP/EVP) Settings
ifm: Immediate Forward / SRM Settings
logger: Data Logger Settings
stat: Print stack stats using filters
stats: Print stack stats using a key
find: Case insensitive search using filters
version: Print muci version
```

Note: The above list is from an RSU. The OBU topics will be different.

To see the capabilities for a topic, just type **muci <topic>**, for example:

```
root@ITS-RS4-M-1001848:~# muci srm
Usage:
muci srm get <key>
muci srm set <key> <value> [<key> <value>]
muci srm del <key>
muci srm find <filters>
muci srm create
muci srm add <priority|preemption> <requestId> <inboundLanes(s)> [<outboundLanes(s)>]
```

In general, **get, set, del** and **find** are available for all config-based topics. Some topics have additional, topic-specific operations.

Forward and reverse command syntax

muci its find searches in the its "namespace" only, therefore the results are not prefixed with "its." This is called the "forward" command syntax. Of course, you can use other topics instead of **its**.

muci find searches globally (meaning all the topics handled by **muci**), therefore the results are prefixed. This is called the "reverse" or "Major" command syntax. The reverse command syntax has some limitations but provides an easy way of finding a key and copy-pasting it for a subsequent **muci set** operation:

```
root@ITS-RS4-M-1001848:~# muci find sensor
saf.native.sensorSharing.enable = false
saf.native.sensorSharing.excludeV2xSendingObjects = false
saf.native.sensorSharing.mode = "Psm"
cff.sensorAdapters.camera.detectionAreas[].vertices[].lat = 0
cff.sensorAdapters.camera.detectionAreas[].vertices[].lon = 0
cff.sensorAdapters.camera.direction = 0
cff.sensorAdapters.camera.fov.vertices[].lat = 0
cff.sensorAdapters.camera.fov.vertices[].lon = 0
cff.sensorAdapters.camera.position.lat = 60.182832
cff.sensorAdapters.camera.position.lon = 24.955525
cff.sensorAdapters.udpAdapter.enable = true
cff.sensorAdapters.udpAdapter.port = 12321

root@ITS-RS4-M-1001848:~# muci set saf.native.sensorSharing.enable true
```

Accessing stack counters

Stack counters are very frequently used during all kinds of setup and debug activities.

Accessing the stack counters was relatively easy on y18:

```
root@ITS-OB4-M-1001981:~# unplugged-rt-status-gen 127.0.0.1|grep -i bsm
...
  "FAC sent BSM": "698303",
...
```

This no longer works on y20, because the output of the corresponding script is json, and the counter names and values are printed on different lines, so grep won't find anything useful.

```
root@ITS-OB4-M-1001981:~#v2x-status-json-gen
...
"tx": {
"usBsm": {
          "txPacket": 0
        },
...
```

Therefore, **muci** offers a simple way to access the counters (and all status items):

muci stat and **muci stats** are two topics that do not handle config files, but provide a user interface to the stack counters generated by `v2x-status-json-gen`.

muci stat <filters> accepts filters (similar to **muci find**)

muci stats <key> accepts a key (similar to **muci get**)

```
root@ITS-RS4-M-1001848:~# muci stat rx bsm us
statistics.facility.rx.usBsm.hasSubscribers = 1
statistics.facility.rx.usBsm.rxDroppedPacket = 0
statistics.facility.rx.usBsm.rxPacket = 16428
```

```
root@ITS-RS4-M-1001848:~# muci stats statistics.facility.rx.usBsm.rxPacket
statistics.facility.rx.usBsm.rxPacket = 20492
```

User guide

This section builds on top of the Quick Start Guide and provides additional information.

Key notation

`muci` uses a dotted notation to reference values in a json structure. For example:

```
mapping.intersectionId
```

is a key that references the value stored as:

```
"mapping": {
        "intersectionId": 42786,
        ...
}
```

Similarly,

```
mapping.signalGroups[0].signalGroupId
```

references the first value inside an array item:

```
"mapping": {
        "signalGroups": [
            {
                "signalGroupId": 1,
          ...
            }
        ]
}
```

Discovering keys and supported values

If you are unsure about a certain key or the possible values, you can use **muci set** without a value to discover the possible keys and values.

muci set will show:

- Current value (if available)
- Descriptions
- Value types
- Enum lists
- Members of non-existent array items that can be created

Examples:

```
root@ITS-RS4-M-1001848:~# muci its set capture.address
capture.address = "127.0.0.1"
Type: string
The IPv4 address of the remote server receiving the captured data.
```

```
root@ITS-RS4-M-1001848:~# muci srm set request.priority.type
request.priority.type = "Intelight"
Type: string
Type of the priority request.
Valid values for request.priority.type:
Ntcip1211v02
Intelight
TrafficwareV2
```

```
root@ITS-RS4-M-1001848:~# muci tlc set mapping.signalGroups
mapping.signalGroups[].clearanceLength = 0
mapping.signalGroups[].enable = true
mapping.signalGroups[].greenLength = 0
mapping.signalGroups[].mergeTimings = true
mapping.signalGroups[].preMovementLength = 0
mapping.signalGroups[].primaryDataIndex
mapping.signalGroups[].primaryDataSource = "Phase"
mapping.signalGroups[].primaryDataTrafficHeadType
mapping.signalGroups[].redLength = 0
mapping.signalGroups[].secondaryDataIndex = 1
mapping.signalGroups[].secondaryDataSource = "Phase"
mapping.signalGroups[].secondaryDataTrafficHeadType = "Unavailable"
mapping.signalGroups[].signalGroupId
mapping.signalGroups[].statePrediction = false
Array of signal groups sent in SPaT.
```

*Note: if the array does not exist, **muci set** will show all possible array items.*

muci find will also show members of non-existent array items, but not the other attributes.

Setting a value

Once you know the key, you can set the value using multiple forward and reverse command syntax options:

Syntax	Example
muci <topic> set <key> <value>	`muci its set capture.port 1234`
muci set <topic>.<key> <value>	`muci set its.capture.port 1234`
muci <topic> set <key> = <value>	`muci its set capture.port = 1234`
muci set <topic>.<key> = <value>	`muci set its.capture.port = 1234`
muci <topic> set <key>=<value>	`muci its set capture.port=1234`
muci set <topic>.<key>=<value>	`muci set its.capture.port=1234`

String values are accepted with or without quotation marks

```
muci set its.capture.address "127.0.0.1"
```

is equivalent to

```
muci set its.capture.address 127.0.0.1
```

Default values

Some default values come from the schema, so removing the value from the config json has the same effect as specifying the default value.

```
root@ITS-OB4-M-1001981:~#muci its set capture.enable false
```

Note: this is the default value, so the key is removed from its.json

```
root@ITS-OB4-M-1001981:~#muci its set capture.port 1234
```

*Note: this is **not** the default value, so the key is added to its.json*

Validation

When changing a value, the config will only be modified (written) if the resulting json document passes schema validation:

```
root@ITS-OB4-M-1001981:~#muci its set capture.address 292.168.1.1
Error: Invalid value
```

Setting multiple values

Sometimes it is necessary to set multiple values in a single command. For example, the schema may require more than one value in an array item. This can be achieved by using multiple **<key> <value>** pairs. This feature is only available using the forward syntax without "=".

```
root@ITS-RS4-M-1001848:~# muci tlc set mapping.signalGroups[].signalGroupId 1
Error: Invalid value!
root@ITS-OB4-M-1001981:~# muci tlc set mapping.signalGroups[].signalGroupId 1 \
mapping.signalGroups[-1].primaryDataIndex 1 \
mapping.signalGroups[-1].primaryDataTrafficHeadType Protected
```

Note: the first key contains [] so it will add a new array item while subsequent keys reference the newly added item using the index [-1]. See array indexing below.

Handling arrays

Array addressing

- [0] references the first array item
- [-1] references the last array item
- In case of value assignment, [] adds an array item
- When querying the config, [] indicates an empty or non-existent array or unspecified array index.

Examples:

```
root@ITS-RS4-M-1001848:~# muci tlc find sig
mapping.signalGroups[].clearanceLength = 0
mapping.signalGroups[].enable = true
mapping.signalGroups[].greenLength = 0
mapping.signalGroups[].mergeTimings = true
mapping.signalGroups[].preMovementLength = 0
mapping.signalGroups[].primaryDataIndex =
mapping.signalGroups[].primaryDataSource = "Phase"
mapping.signalGroups[].primaryDataTrafficHeadType =
mapping.signalGroups[].redLength = 0
mapping.signalGroups[].secondaryDataIndex = 1
mapping.signalGroups[].secondaryDataSource = "Phase"
mapping.signalGroups[].secondaryDataTrafficHeadType = "Unavailable"
mapping.signalGroups[].signalGroupId =
mapping.signalGroups[].statePrediction = false
```

*Note: the array does not exist, but **muci find** shows the structure of the possible array items, along with default values where available.*

```
root@ITS-RS4-M-1001848:~# muci tlc get mapping.signalGroups[].signalGroupId
mapping.signalGroups[0].signalGroupId = 1
mapping.signalGroups[1].signalGroupId = 2
mapping.signalGroups[2].signalGroupId = 3
mapping.signalGroups[3].signalGroupId = 4
mapping.signalGroups[4].signalGroupId = 5
mapping.signalGroups[5].signalGroupId = 6
mapping.signalGroups[6].signalGroupId = 7
mapping.signalGroups[7].signalGroupId = 8
```

Note: for an existing array, [] acts as a wildcard in the key.

Adding a new array item

A new array item can be added using **muci set** combined with one of the array item addressing methods. If the specified item already exists, it will be overwritten and no item will be added. If the index references the item following the last existing item, a new item will be added.

```
root@ITS-RS4-M-1001848:~# muci its get logging.debugComponents #array is empty
root@ITS-RS4-M-1001848:~# muci its set logging.debugComponents[0] psm #add
root@ITS-RS4-M-1001848:~# muci its set logging.debugComponents[0] map #overwrite
root@ITS-RS4-M-1001848:~# muci its set logging.debugComponents[] spat #add
root@ITS-RS4-M-1001848:~# muci its set logging.debugComponents[2] was #add
root@ITS-RS4-M-1001848:~# muci its set logging.debugComponents[-1] bsm #overwrite
root@ITS-RS4-M-1001848:~# muci its set logging.debugComponents[4] tim #fails to add
Error: Invalid key!
root@ITS-RS4-M-1001848:~# muci its get logging.debugComponents
logging.debugComponents[0] = "map"
logging.debugComponents[1] = "spat"
logging.debugComponents[2] = "bsm"
```

Removing an array item

An item can be removed using **muci del <key>** assuming that the key references an array item.

```
root@ITS-RS4-M-1001848:~# muci its del logging.debugComponents[1]
root@ITS-RS4-M-1001848:~# muci its get logging.debugComponents
logging.debugComponents[0] = "map"
logging.debugComponents[1] = "bsm"
```

Removing an entire array

An item can be removed using **muci del <key>** assuming that the key references an entire array.

```
root@ITS-RS4-M-1001848:~# muci its del logging.debugComponents
root@ITS-RS4-M-1001848:~# muci its get logging.debugComponents
root@ITS-RS4-M-1001848:~#
```

Deleting non-array keys

muci del can be used to remove keys from the json config. In this case, default values from the schema will take effect. Use with caution.

Handling data logger configs

Most topics have just one json config file, and the name is fixed (and hidden from the user). The data logger facility, however, supports multiple config files and the user has to name them. Therefore, the data logger topic has a different command syntax:

```
root@ITS-RS4-M-1001848:~# muci logger
Usage:
muci logger <filename> get <key>
muci logger <filename> set <key> <value> [<key> <value>]
muci logger <filename> del <key>
muci logger <filename> create
```

The **<filename>** option should not include the path or the extension.

muci logger <filename> create will create a skeleton config file that satisfies the schema. (An empty json would fail schema validation.)

```
root@ITS-RS4-M-1001848:~# muci logger test create
root@ITS-RS4-M-1001848:~# muci logger test create
File /rwdata/v2x_configs/data_logger_ftw/test.json already exists
root@ITS-RS4-M-1001848:~# muci logger test get
enabled = false
out.socket.destHost = "127.0.0.1"
out.socket.destPort = 1234
out.socket.protocol = "udp"
source[0] = "Wsmp"
version = 5
```

Handling SNMP user config files

```
root@ITS-RS4-M-1001848:~# muci snmp
Usage:
muci snmp <filename> get <key>
muci snmp <filename> set <key> <value> [<key> <value>]
muci snmp <filename> del <key>
muci snmp <filename> create <user> <pwd>
```

Similarly to data logger configs, the file name must be specified. The **create** command requires a username and a password.

Custom commands

In order to provide an easy way to perform some frequently used tasks, **muci** offers application-specific command extensions for certain topics.

For example, a single TLC signal ID mapping requires 5-7 parameters. While it is possible to set all of these parameters using **muci set**, it would be very cumbersome, error-prone and confusing for the user, so **muci** offers a single command to create a signal ID mapping (see below).

Customs commands can only be used with the forward syntax.

Use **muci <topic>** to explore the available commands for each topic.

TLC custom commands

```
root@ITS-RS4-M-1001848:~# muci tlc
Usage:
muci tlc get <key>
muci tlc set <key> <value> [<key> <value>]
muci tlc del <key>
muci tlc find <filters>
muci tlc add <signal id> <Phase|Overlap> <Primary source> <Permissive|Protected>
[<Phase|Overlap> <Secondary source> <Permissive|Protected>]
muci tlc list
muci tlc rem <signal id>
```

A typical 8-way intersection mapping can be set up using the following set of **muci tlc add** commands:

```
root@ITS-RS4-M-1001848:~# muci tlc add 1 Phase 1 Protected Phase 6 Permissive
root@ITS-RS4-M-1001848:~# muci tlc add 2 Phase 2 Permissive
root@ITS-RS4-M-1001848:~# muci tlc add 3 Phase 3 Protected Phase 8 Permissive
root@ITS-RS4-M-1001848:~# muci tlc add 4 Phase 4 Permissive
root@ITS-RS4-M-1001848:~# muci tlc add 5 Phase 5 Protected Phase 2 Permissive
root@ITS-RS4-M-1001848:~# muci tlc add 6 Phase 6 Permissive
root@ITS-RS4-M-1001848:~# muci tlc add 7 Phase 7 Protected Phase 4 Permissive
root@ITS-RS4-M-1001848:~# muci tlc add 8 Phase 8 Permissive
```

In order to save on typing, the keywords can be abbreviated as long as they are unambiguous:

muci tlc add 1 p 1 pr p 6 pe

is equivalent to

muci tlc add 1 Phase 1 Protected Phase 6 Permissive

The signal ID is considered a unique key, therefore if a signal group exists with the same ID, it will be overwritten.

The mapping can be inspected using **muci tlc list**:

```
root@ITS-RS4-M-1001848:~# muci tlc list
1 Phase 1 Protected Phase 6 Permissive
2 Phase 2 Permissive
3 Phase 3 Protected Phase 8 Permissive
4 Phase 4 Permissive
5 Phase 5 Protected Phase 2 Permissive
6 Phase 6 Permissive
7 Phase 7 Protected Phase 4 Permissive
8 Phase 8 Permissive
```

A mapping can be removed using **muci tlc rem**:

```
root@ITS-RS4-M-1001848:~# muci tlc rem 8
root@ITS-RS4-M-1001848:~# muci tlc list
1 Phase 1 Protected Phase 6 Permissive
2 Phase 2 Permissive
3 Phase 3 Protected Phase 8 Permissive
4 Phase 4 Permissive
5 Phase 5 Protected Phase 2 Permissive
6 Phase 6 Permissive
7 Phase 7 Protected Phase 4 Permissive
```

SRM processor custom commands

```
root@ITS-RS4-M-1001848:~# muci srm
Usage:
muci srm get <key>
muci srm set <key> <value> [<key> <value>]
muci srm del <key>
muci srm find <filters>
muci srm create
muci srm list
muci srm add <priority|preemption> <requestId> <inboundLanes(s)> [<outboundLanes(s)>]
```

muci srm add offers an easy way to set up lane assignments.

The tricky part is that certain fields must be set up before the command can be used. **muci** tries to be helpful here by offering command hints:

```
root@ITS-RS4-M-1001848:~# muci srm add pri 4 21,22
Set priority type first
muci srm set request.priority.type
root@ITS-RS4-M-1001848:~# muci srm set request.priority.type
request.priority.type
Type: string
Type of the priority request.
Valid values for request.priority.type:
Ntcip1211v02
Intelight
root@ITS-RS4-M-1001848:~# muci srm set request.priority.type Intelight
root@ITS-RS4-M-1001848:~# muci srm add pri 4 21,22
Set network address first
muci srm set request.priority.intelight.network.address
root@ITS-RS4-M-1001848:~# muci srm set request.priority.intelight.network.address
1.1.1.1
root@ITS-RS4-M-1001848:~# muci srm add pri 4 21,22
root@ITS-RS4-M-1001848:~# muci srm get request.priority.intelight.laneAssignments
request.priority.intelight.laneAssignments[0].inBoundLanes.laneIds[0] = 21
request.priority.intelight.laneAssignments[0].inBoundLanes.laneIds[1] = 22
request.priority.intelight.laneAssignments[0].requestId = 4
```

muci srm list provides an easy-to-read output:

```
root@ITS-RS4-M-1001848:~# muci srm list
Priority requests over Intelight protocol:
1 16,17,18,19
2 4,5,6
3 9,10,11
4 21,22
Preemption requests over Ntcip1202 protocol:
6 21,22
3 16,17,18,19
4 4,5,6
5 9,10,11
```

Sample commands

The list below attempts to offer a list of sample commands that encompass most **muci** features. This is not a script – it is a walkthrough where the user is expected to issue commands one-by-one and observe command output. None of these commands should fail, but some will produce an obvious error message.

```
muci
muci its
muci its get
muci its get capture.enable
muci its get capture
muci its set capture.enable true
muci its set capture.address 127.0.0.1
muci its get capture
muci its find port
muci find port
muci tlc set tlc.listenPort 1
muci tlc get tlc.listenPort
muci tlc set tlc.listenPort=2
muci tlc get tlc.listenPort
muci tlc set tlc.listenPort = 3
```

```
muci tlc get tlc.listenPort
muci set tlc.tlc.listenPort 4
muci tlc get tlc.listenPort
muci set tlc.tlc.listenPort=5
muci tlc get tlc.listenPort
muci set tlc.tlc.listenPort = 5001
muci tlc get tlc.listenPort
muci tlc set tlc.protocol
muci tlc
muci tlc del mapping.signalGroups
muci tlc get mapping.signalGroups
muci tlc set mapping.signalGroups
muci tlc set mapping.signalGroups[].signalGroupId 1
mapping.signalGroups[-1].primaryDataIndex 1 mapping.signalGroups[-
1].primaryDataTrafficHeadType Protected
muci tlc list
muci tlc get mapping.signalGroups
muci tlc add 2 "Phase" 1 "Protected" "Phase" 6 "Permissive"
muci tlc list
muci tlc add 3 Phase 2 Permissive
muci tlc list
muci tlc add 4 p 4 pe o 5 pr
muci tlc list
muci tlc get mapping.signalGroups
muci tlc get mapping.signalGroups[]
muci tlc get mapping.signalGroups[0]
muci tlc get mapping.signalGroups[-1].signalGroupId
muci tlc get mapping.signalGroups[].signalGroupId
muci tlc del mapping.signalGroups[1]
muci tlc list
muci tlc rem 3
muci tlc list
rm /rwdata/v2x_configs/srm_processor.json
muci srm get
muci srm
muci srm create
muci srm get
muci srm add priority 1 11,12
muci srm set request.priority.type
muci srm set request.priority.type Intelight
muci srm add priority 1 11,12
muci srm set request.priority.intelight.network.address
muci srm set request.priority.intelight.network.address 1.1.1.1
muci srm set request.priority.intelight.network.address "2.2.2.2"
muci srm add priority 1 11,12
muci srm add priority 2 13,14 15,16
muci srm get request.priority
muci srm list
muci find 7942
muci find senso
muci set saf.native.sensorSharing.enable true
rm /rwdata/v2x_configs/data_logger_ftw/test.json
```

```
muci logger test get
muci logger test create
muci logger test get
muci logger test create
muci logger test set
muci logger test set filters[].psid 11
muci logger test get
muci stats statistics.facility.rx.usBsm.rxPacket
muci stat rx bsm us
rm /rwdata/etc/snmpd-data/v3users/test.json
muci snmp test create
muci snmp test create alma almafa
muci snmp test create alma almafa
muci snmp test get
muci version
```