**File Formats for the Sort Segment of the System Support Mapping Processing Sequence**

*Steve Chall*

*2017/07/24*

**1. Intro**

This document describes several file formats used for the *sort* segment of the System Support Mapping (SSM) processing sequence.

It does not specify the following file formats also generated by and used in the processing sequence: the SSM .json file format (see <https://github.com/steve9000gi/ssm>), the BLM.csv and CBLM.csv file formats (discussed in <https://github.com/steve9000gi/binary-link-matrix> and <https://github.com/steve9000gi/AddCodesToBLM>, respectively) or the CM.csv and CPM.csv formats (both in <https://github.com/steve9000gi/create_code_matrices>).

The full SSM processing sequence is demonstrated by example in <https://github.com/steve9000gi/ssm/blob/master/SSM_Processing_Sequence.txt>.

In this document any string in < angle brackets> stands in for some “quoted string” whose actual contents in any particular instance will be dependent on the context of a specific set of originating SSMs. An ellipsis (“…”) means “etc.” or “and more of the same”.

**2. blm.R output**

In addition to a set of BLM.csv files, the R script blm.R outputs also outputs several files any of which may be used as drag-and-drop inputs for the sort website (<http://syssci.renci.org/sort>).

**2.1. aggregated.txt**

*aggregated.txt* is, as its extension implies, a text file, and it displays all the node texts from all the SSMs in the blm.R input directory, organized by SSM node type, as follows:

ROLES:

<role node text 0>

<role node text 1>

…

<role node text nroles>

*“\n” [blank line]*

RESPONSIBILITIES:

<responsibilities node text 0>

<responsibilities node text 1>

…

<responsibilities node text nresponsibilities>

*“\n”*

NEEDS:

<needs node text 0>

<needs node text 1>

…

<needs node text nneeds>

*“\n”*

…

and similarly through the other ring names RESOURCES, WISHES, and TEXTS, where there are 0 to nroles ROLES node texts, 0 to nresponsibilities RESPONSIBILITIES node texts, etc.

**2.2. aggregated.json**

{

"unsorted": {

"NEEDS": {

"textItems": [

{

"text": <needs node text 0>

},

{

"text": <needs node text 1>

},

...

{

"text": <needs node text nneeds>

}

]

},

"RESOURCES": {

"textItems": [

{

"text": <resources node text 0>

},

{

"text": <resources node text 1>

},

...

{

"text": <resources node text nresources>

}

]

},

"RESPONSIBILITIES": {

"textItems": [

{

"text": <responsibilities node text 0>

},

{

"text": <responsibilities node text 1>

},

...

{

"text": <responsibilities node text nresponsibilities>

}

]

},

"ROLES": {...},

"TEXTS": {...},

"WISHES": {…}

}

}

Note that ring order doesn’t matter. In the .json example above, the rings are ordered alphabetically. In the text example, they’re ordered according to SSM position. Either or another order is ok for both .json and .txt formats.

**2.3 Other blm.R .json output files**

There are also separate .json files output by blm.R for each of the rings, to facilitate sorting texts for one ring (i.e., node) type at a time. As a representative example, the format for *NEEDS.json* is as follows:

{

"unsorted": {

"NEEDS": {

"textItems": [

{

"text": <needs node text 0>

},

{

"text": <needs node text 1>

},

...

{

"text": <needs node text nneeds>

}

]

}

}

}

where there are 0 to nneeds *Needs* text items.

**3. Sort file formats: input and output**

The *sort* website will accept as input any of the file inputs described in Section 2 above. Note that for all the .json formats in section 2 there is only one top-level key-value pair inside the outermost {}:

{“unsorted”: {…}}

This is because blm.R outputs only unsorted node text items, organized by ring.

The full *sort* input-output format has a second top-level key-value pair – key: “sorted” – whose lower-level values are organized by (i.e., sorted according to) *codes* assigned during the sort process, so that the top-level structure looks like this:

{“sorted”: […], “unsorted”: {…}}

Each element of the “sorted” array is an object representing a *code* plus the text items associated with that code. Each element of the “unsorted” object is an object representing a ring name (i.e., SSM node type: role, need, etc.) plus an array of the text items belonging to that ring.

The full *sort* input-output format includes several additional key-value pairs for each text object, both for sorted and unsorted items. The values associated with keys “dataIndex” and “displayedText” define appearance and positioning of text when displayed on the *sort* website: “displayedText” defines how an item looks when it’s displayed in the “unsorted” column, i.e., with the addition of a number <n> in front of the text item followed by “.  “ and the text item. The value associated with “dataIndex” determines where a text item is placed in the unsorted column, or where it is reinserted if it is later removed from one of the coded boxes. The “id” key is used to maintain unique id values and the DOM text element is assigned that id. “text” is used to hold the core text value – the actual node text item as is in the originating SSM – whether or not it’s numbered in the unsorted list.

Under “unsorted,” ring names (“ROLES”, “NEEDS”, etc.) receive additional fields in *sort* input-output format output: “dataIndex”, again for positioning in the unsorted list; “id”, for assignment in the DOM.

“sorted” text objects will have these same additional fields as unsorted text items, and for the same reasons: “dataIndex”, “text”, “displayedText”, and “id”.

Under “sorted”, *codes* fulfill an organizing function somewhat like the ring names under “unsorted”, in that an array of “textItems” belongs to each. However, codes provide the titles for the boxes into which text Items are sorted, and they have these fields: “title”, the box name and ultimately the code itself; “xform” which determines where the corresponding box is placed in the *sort* window.

These fields are optional for unsorted *sort* input, and will be provided by *sort*: “dataIndex” and “id” (for “textItems” sorted and unsorted, and ring names, the latter under “unsorted” only); “displayedText” (for all “textItems”); “title” and “xform” (for code names, which are by definition “sorted” only). Once some items are sorted, all fields are required (and provided in *sort* .json output) and will be used when a partially or fully sorted file is reopened in *sort*.

**4. AddCodesToBLM.R input**

This same .json full file format is used for input into AddCodesToBLM.R for the first post-sort step in the SSM file processing sequence. At this time, the *sort*-specific elements are ignored: “id”, “xform”, “displayedText”, “dataIndex” for all sorted elements (textItems and codes), and everything under “unsorted.” Henceforward, we are only interested in the codes that have been assigned, and the text item[s] associated with each.

**5. Example of fully-populated .json *sort* file format**

Let there be a (very small) set of SSMs input into blm.R, and let the full aggregated.txt file display the following:

ROLES:

Role0

Role1

NEEDS:

Need0

Need1

Need2

Need3

RESPONSIBILITIES:

Resp0

Resp1

RESOURCES:

Resource0

WISHES:

Wish1

TEXTS:

Text0

Then let this dataset be partially sorted as follows (this is not a formatted file. Rather, it’s intended to demonstrate in an easily understood manner the elements from which a formatted file will be constituted):

*Unsorted:*

ROLES:

NEEDS:

Need1

RESPONSIBILITIES:

Resp0

Resp1

RESOURCES:

WISHES:

Wish1

TEXTS:

*Sorted:*

Code0:

Role0

Role1

Code1:

Need0

Need2

Need3

Code3:

Resource0

Text0

(Note that all the roles (“Role0” and “Role1”), all resources (“Resource0”), and all texts (“Text0”) have been sorted, and that there remain a total of four unsorted text items.)

If such a dataset were exported as .json from *sort*, the resulting file would look similar to the following:

{

"sorted": [

{

"textItems": [

{

"dataIndex": "1",

"displayedText": "1. Role0",

"id": "id18",

"text": "Role0"

},

{

"dataIndex": "2",

"displayedText": "2. Role1",

"id": "id19",

"text": "Role1"

}

],

"title": "Code0",

"xform": "translate(189.75,14.16)"

},

{

"textItems": [

{

"dataIndex": "4",

"displayedText": "3. Need0",

"id": "id21",

"text": "Need0"

},

{

"dataIndex": "6",

"displayedText": "5. Need2",

"id": "id23",

"text": "Need2"

},

{

"dataIndex": "7",

"displayedText": "6. Need3",

"id": "id24",

"text": "Need3"

}

],

"title": "Code1",

"xform": "translate(194.75,108.16)"

},

{

"textItems": [

{

"dataIndex": "12",

"displayedText": "9. Resource0",

"id": "id29",

"text": "Resource0"

},

{

"dataIndex": "16",

"displayedText": "11. Text0",

"id": "id33",

"text": "Text0"

}

],

"title": "Code2",

"xform": "translate(196.75,206.16)"

}

],

"unsorted": {

"NEEDS:": {

"dataIndex": "3",

"id": "id20",

"textItems": [

{

"dataIndex": "5",

"displayedText": "4. Need1",

"id": "id22",

"text": "Need1"

}

]

},

"RESOURCES:": {

"dataIndex": "11",

"id": "id28",

"textItems": []

},

"RESPONSIBILITIES:": {

"dataIndex": "8",

"id": "id25",

"textItems": [

{

"dataIndex": "9",

"displayedText": "7. Resp0",

"id": "id26",

"text": "Resp0"

},

{

"dataIndex": "10",

"displayedText": "8. Resp1",

"id": "id27",

"text": "Resp1"

}

]

},

"ROLES:": {

"dataIndex": "0",

"id": "id17",

"textItems": []

},

"TEXTS:": {

"dataIndex": "15",

"id": "id32",

"textItems": []

},

"WISHES:": {

"dataIndex": "13",

"id": "id30",

"textItems": [

{

"dataIndex": "14",

"displayedText": "10. Wish1",

"id": "id31",

"text": "Wish1"

}

]

}

}

}