The following files are part of a larger data set.

1. [**0-sys-components.xml**](file:///C:\Users\amohamed\Downloads\Access%20to%20Resources.html\0-sys-components.xml)
2. [**comp-0-backend.main.main.c.xml**](file:///C:\Users\amohamed\Downloads\Access%20to%20Resources.html\comp-0-backend.main.main.c.xml)
3. [**comp-100-backend.commands.explain.c.xml**](file:///C:\Users\amohamed\Downloads\Access%20to%20Resources.html\comp-0-backend.main.main.c.xml)
4. [**comp-516-backend.utils.misc.ps\_status.c.xml**](file:///C:\Users\amohamed\Downloads\Access%20to%20Resources.html\comp-516-backend.utils.misc.ps_status.c.xml)

These are XML files that include a part of an example software architecture for experimental use within our project. The complete system includes around 1200 components. N ow, we practice using (Comp 0, comp 100 and Comp 516). The first link above (0-sys-components) lists all the components by name as well as some related info. The following is interpretation to some of the terminology in this dataset.

## Main file (0-sys-components) format

* **Component Arch\_profile** means the component name
* **File** means Component file location on the hard disk
* **Num\_CICBs** means the number of blocks (or sections, i.e., function definitions)  of the component
* **Name** is similar to Component Arch\_profile above
* **Id** means the Id of the component

## Component files (Comp 0, 100, and 516) format

# Every component includes a number of blocks (or sections). For example, comp 0 includes 6 blocks. All the blocks are listed in the component file with details about every block.  For every block, you may find some of the following  details.

* **>>>the following data items are repeated with every block in the component**
* **CICB Name** means the Block (section) name
* **Id** means the Id of the block
* **<provide ... >  details  </Provide>:** details of connections INTO the component. (You may discard this part, unless you need it)
* **<Require ... >  details  </Require >:** details of connections OUT OF the component. (You may discard this part, unless you need it )
* **Point\_Block** means the start of block details.
* **Code\_Loc** means code location (preliminary data)
* **Braced** means has parenthesis or not (preliminary data)
* **Num\_points** means, the number of main sub-blocks in this block.
* **>>>After this point sub-blocks are listed,** data items are also described as follows
  + **Date Code\_Loc** means code location of the sub-block (preliminary data)
  + **Data\_owner**means the type of the sub-block
  + **Variable Name** means the same (Variable Name)
  + **Home\_block** means location where the variable/or call exists
  + **Accessd\_block** means location where the variable/or call are accessed from.
  + **Access**means type of access
  + **Call Name means** connection name

## Data Transformation

**To transform this raw data into CSV or JSON, you need to use some library that reads/parses XML files.**

**Example online converters from XML to CSV or JSON**

**Example XML parsing in Python language**

* **Using ElementTree Module,** [**XML Parser in Python**](https://www.geeksforgeeks.org/xml-parsing-python/)
* **Using MiniDom module,** [**Reading and Writing XML Files in Python**](https://stackabuse.com/reading-and-writing-xml-files-in-python/)

**Example XML parsing in Java language**

* **Using JDom Parser Module,** [**Java DOM Parser - Parse XML Document**](https://www.tutorialspoint.com/java_xml/java_dom_parse_document.htm)

You can also find some JavaScript Library that parses XML files into JSON.

You can also write some code to read XML as text data and create the CSV/JSON files by your code. I have some code written in Java that does this already, however, I prefer that we make this the last solution since it is better to find a more creative solution for this conversion.

## Example data model for tree structure

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | |
| **Elm\_Id** | **Element** | **Elm\_Type** | **Parent\_Id** | **f**ield**\_4** | **Field\_5** | **...** | **Field\_n** |
| 10001 | root | System |  |  |  | ... |  |
| 10002 | main.c | Component | 10001 | Num\_Blocks: 6 | Path: ~~~ | ... | ... |
| 10003 | Master | Block | 10002 | Code\_loc:(48,1)(~ | Require 0 |  | Provide 0 |
| 10004 | main | Block | 10002 | Code\_Loc:(61,2)( | Require 6 |  | Provide 5 |
| 10005 | startup\_hacks | Call | 10002 | ~ | ~ | ... | ~ |
|  |  |  |  |  |  |  |  |

You don't have to extract everything in these files, just capture some easy parts for now that make the job faster. The only challenging part is [1>> solution 1] converting the XML into an easier CSV/JSON format.

We also can try to [1>> solution 2] create a D3 function (like indentedtree() ) that reads directly from XML/ or JSON (obtained from the given XML).

You may investigate the two solutions and discuss with me your findings.