# Dynamic Table of Contexts Commons Step-by-Step Tutorial

By Rachel Milio

This page will explain how (and why) to use the [Dynamic Table of Contexts](https://dtoc.lincsproject.ca/) to create a digital edition of a text (using the DToC Commons application). This guide is not entirely linear, and I recommend that you navigate back and forth, especially in between steps 2 and 3, in order to test your DTOC as you work on it.

To start, you must have:

1. A Github/Gitlab account
2. An XML document tagged according to the TEI standards

NOTE: Not all documents are conducive to the Dynamic Table of Contexts.

In order to determine whether your document would be a good fit for creating a DToC edition, first consider whether a standard table of contents would facilitate navigation of your document. DToC presents your document with a table of contents made up of smaller sections in the larger document (be it chapters of a book, acts of a play, or shorter documents compiled together). While shorter, one-part documents can still be enriched by the use of DToC, its full capability assists with the navigation of longer documents.

Second, consider the TEI tags and elements of your document. DToC allows you to navigate through a document via tags, so an entity-rich document would work best. The way DToC presents tags can be curated and specified depending on what information seems most relevant to you.

Lastly, consider the concepts present in your document(s). DToC also allows for the navigation of a document based on index terms, so a document rich with concepts (ideally beyond the person and place entities that would already be captured via tags) would be an appropriate input.

The example document used in this tutorial is the [Memorable Masque of Lincoln’s Inn and Middle Temple](https://dtoc.lincsproject.ca/view?document=https%3A%2F%2Fraw.githubusercontent.com%2FREEDLondon%2Fcivic%2Fmain%2Fmemorablemasque_dtoc.json) by George Chapman. Here is [the XML file used](https://raw.githubusercontent.com/rachelmilio/civic/main/memorablemasque_1.xml), and here is [the JSON file](https://raw.githubusercontent.com/REEDLondon/civic/main/memorablemasque_dtoc.json).

DToC’s table of contents is used to differentiate the elements of the masque’s front matter (such as performance descriptions and cast list) from the masque’s script. The tags are used to show the lines of each character in the masque text, as well as the songs and stage directions. The index is used to show more conceptual elements, such as allusions to classical mythology, references to contemporary figures, or dramaturgical descriptions such as costumes and sets.

### Step 1: Properly mark-up an XML document

In order to create a DToC, you will first need to ensure your XML file is marked-up correctly.

Structurally, this requires the document to be broken into sections, denoted by the <div> tag. The <div> tags can also benefit from @type attributes, such as <div type=‘chapter’>. Each <div> must also have a title, stored in the <head> tag. I recommend also using the <p> tag within each <div> to denote paragraphs for prose (<ab> (anonymous block) would also work) or <lg> (line group) for verse.

Following structural markup, the document should be semantically tagged. These tags are dependent on your editorial choices (what information do I want to capture and present in this document?). The [TEI Guidelines](https://www.tei-c.org/release/doc/tei-p5-doc/en/html/REF-ELEMENTS.html#) are a helpful resource for determining what tags to use, although tags such as <persName> and <placeName> are a solid starting place if you’re entirely new to TEI. For longer sections of note (for example, references to classical mythology in the *Memorable Masque*), consider using the <span> tag (for example, <span type=‘classical\_allusion’> or <span type=‘literary\_allusion’>. While you’re tagging, think ahead to your potential index and list of displayed tags. While you could tag literally every word in your document, this probably isn’t the best use of your time.

If you’d like to use element attributes (for example, using @ref attributes to link your person/place elements to a knowledge base such as Wikidata), you are encouraged to do so (although this is not necessary for a working DToC). I would recommend the @key attribute or the @ana attribute to clarify exactly who or what an element is referring to. The sentence “Her majesty visited the Tower” could become “<persName key=‘Queen Elizabeth I’ ref=‘https://www.wikidata.org/wiki/Q7207’>Her majesty</persName> visited <placeName key=‘Tower of London’ ref=‘<https://www.wikidata.org/wiki/Q62378>’>the Tower</placeName>.”

Keep your keys consistent! If you use “Queen Elizabeth I” as your key in one place, don’t use “Elizabeth Tudor”, “Elizabeth I”, or “Queen Elizabeth” elsewhere.

Once you’ve tagged your document to your liking, you will need to add unique XML IDs for any element that you intend to have in your index. These are necessary so that DToC is able to locate your indexed elements within the document. If you are using Oxygen, follow [these instructions](https://cwrc.ca/Documentation/public/DITA_Files-Various_Applications/DToC/AddingXMLidsOxygen.html) to add auto-generated XML IDs to your elements. You could also complete this by hand. I recommend adding unique XML IDs to each paragraph in the <p> tag, as well as other elements that you want to index. A paragraph with an XML ID will look like: <p xml:id="p\_qqk\_241\_41c">.

Once you’ve added unique XML IDs, your document is ready for the next step: the creation of an index!

### Step 2: Create an index

Creating an index in DToC can be a bit labour-intensive, but we already have most of the pieces in place, so it won’t be too difficult.

If you want to bypass creating an index for now, you can move directly to Step 3 to view your XML document as a DToC without an index.

The index can be stored either in its own file or at the bottom of your XML file. In this guide, I will be creating an index within the XML file, but the process is not very different either way.

Your index can be in a <div> tag at the bottom of your document (below the final <div> tag of your text). Give this <div> tag the attribute @type=’index’. You could also store your index in a separate file.

You can create your index as a simple list, or you can break this list down into sublists. For the *Memorable Masque*, my index contains three sublists: Classical Allusions, Contemporary References, and Dramaturgy Descriptions. However, to start, you can just make one simple list to get used to the process.

Within <div @type=’index’>, create a <list>. This will store your index items.

Within the list, each index item will be stored in a <ref> tag nested in an <item> tag (<item><ref>Index Item</ref></item>). The <item> tag is necessary to store anything in a list, while the <ref> tag does the heavy lifting of pointing to the location(s) of the term in the document.

Within each <ref> tag, use the attribute @target to point to the XML ID that corresponds to the index item within the tag. For example, the index item pointing to a reference stored in a <span> tag to the Greek comedian Aristophanes will appear as:

<item><ref target="#span\_qnr\_n55\_n1c">Aristophanes</ref></item>. Note that the @target attribute requires you to add a ‘#’ to the beginning of the XML ID.

If there are multiple instances in the text that refer to the same index reference, these can all be stored within the @target attribute, separated by a space. For instance, multiple references to Princess Elizabeth Stuart can be indexed via:

<item><ref target="#persName\_hvr\_1rs\_k1c #persName\_lvr\_1rs\_k1c #persName\_tvr\_1rs\_k1c #persName\_vvr\_1rs\_k1c">Elizabeth Stuart</ref></item>.

The tedious part of this process is accumulating the different XML IDs to put in the index. I recommend writing them down, or having multiple tabs open, whatever works best for you. If you created your XML IDs by hand, this could be easier as they will be less a random jumble of letters and more a system that makes sense to you.

If you’d like to test your index, I’d recommend skipping ahead to Step 3, in order to begin generating your JSON file and viewing your DToC. Because a DToC is ideally built on files stored on a platform such as Github or Gitlab, you will be able to create your DToC and tweak the underlying files afterwards.

If you’d like to break your index into subcategories, you will create an <item> for each category. Within this <item>, you will put a <ref> element with the title of the category, and another <list> element. Within this <list>, include a <head> restating the title of the category, and store the index items in the same <item><ref target=‘#xml\_id’>Index Item</ref></item> structure.

In the <ref> element for the entire category, you will store all of the XML IDs for the index items in the category. This will be a long entry. In the *Memorable Masque* DToC, an index category appears like this:



### Step 3: Create JSON file

Once you’ve tagged your document and added your index, you are ready to create the JSON file which informs the DToC software on how to interpret and present your document.

JSON is an extension of Javascript which stores data as attribute-data pairs. If you are unfamiliar with JSON, don’t worry! All you need to do is fill out [this template](https://github.com/LEAF-VRE/code_snippets/blob/main/DTOC/dtoc_template.json).

The JSON file will also incorporate XPath, a language that creates paths to navigate an XML document. Don’t stress about this either! Most of what you will need to do with XPath is already in the JSON template.

Your first step is to upload your tagged XML document to Github or a similar platform. This is so that your JSON file can refer to a permanent URL, which will make it possible for anyone to recreate your DToC with your JSON file. On Github, through the button on the top right labeled “Raw,” you are able to generate a link to your XML document.

In your copy of the JSON template, you will insert the raw Github link (or the link to wherever/however else you store the document) into the “inputs” field (replace the text that currently reads “link to raw XML file in github”).

Some of the other fields, such as “editionTitle” and “editionSubtitle” are self-explanatory, and you can put in the title and potential subtitle of your DToC edition into these fields.

The field “documents” tells DTOC which parts of your text make up the body of your edition. If you have properly used <div> tags, this should be fairly simple. The XPath format “//div” tells DTOC to look at the contents of every <div> tag. However, if you have created an index stored in the <div type=’index’> tag, I recommend you do not just use “//div” to avoid displaying your index as text within your edition.

If you used an @type attribute (or a different attribute) such as “chapter”, you can incorporate this into the XPath as “//div[@type=‘chapter’]”.

If you have more than one <div> attribute you want to include, these can be added in with the word “or”, such as “//div[@type=‘transcription’or@type=‘translation’]”.

In the *Memorable Masque* DToC, I use different @type attributes to refer to different sections of my text, leading to an XPath which looks like: “///div[@type='letter'or@type='performance'or@type='set'or@type='castlist'or@type='masque']”.

The next field to fill out is the field “documentTitle”. This is the field DToC uses to determine the title of each <div> section. If you created a <head> element in each <div> to store the title, all you will need to put in this field is “head”. This tells DToC that within the <div> tags you denoted in the “documents” field, the title is stored in the <head> tag.

Next, turn your attention to the “indexDocument” field. If you put your index in your document in a <div type=‘index’> tag, for this field you will input “//div[@type=‘index’]”. Your other option, if your index is stored in a separate file, is to insert a raw Github link, similar to how you completed the “inputs” field.

I’d recommend at this point to upload your JSON file to your public Github repository and test to make sure everything is working as intended. On the [DToC Commons landing page](https://dtoc.lincsproject.ca/), sign in with your Github/Gitlab account. Select either “From the Cloud” to access the file in your Github/Gitlab account, or “From URL” to paste in the raw link to your JSON file. If all goes well, you should have a working DToC!

The next step is to fill out the markup curation field. If you uploaded the template as is and navigate to the Tags tab on the left hand side of your DToC, you should see tags under “People”, “Places”, “Organizations”, and “Objects” (if the corresponding tags are present in your document(s)).

The Tags tab pulls from the “markup” section of the JSON within the “curation” section. Each Tag entry requires an XPath path and a label. For example, the XPath “//persName” is associated with the label “People”.

Currently, this is a functional Tag tab. However, it is much more valuable to finetune the markup section. For instance, if an important person occurs frequently in your document (for example, King James I), and you included an @key attribute in their <persName> element, you can point to the instances of this person within your text. The code for this would, for example, appear like:

{

"xpath": "//persName[@key=‘King James I’]",

"label": "King James I"

}

Any tag within your document is fair game for this type of markup curation. You are not limited to people, places, and organizations. For example, the *Memorable Masque* is a play with character dialogue. I used the Tags tab to display the lines of different characters, which are contained in the <speaker> tag with the attribute @n=‘Character Name’. For instance, the character Plutus appears as:

{

"xpath": "//speaker[@n='Plutus']",

"label": "Plutus"

}

In curating your tags, think about what you can use this section to display that is different from what you present in your index. The index displays concepts and subjects, while the tags are best for displaying a specific category of entities.

I’d like to reiterate here that these steps do not need to be followed linearly, or completed entirely before moving on to the next. DToC Commons needs an XML file, an index (either in the XML file or separately), and a JSON file to function. Your index can have a single item in it and still function, as can your markup curation. As you experiment with building your digital edition with DToC, you may develop new ideas that lead you to tweak your TEI encoding. Because I recommend you store your files on Github/Gitlab, you can easily alter your files when necessary.

If you have any questions or concerns about DToC Commons or the process for creating a DToC edition, feel free to consult [the more extensive DToC documentation](https://cwrc.ca/Documentation/public/DITA_Files-Various_Applications/DToC/OverviewDToC.html) or contact us at \_\_\_ (slack channel? email?).