# Official logo of the Special Interest Group on the Design of Communication

# How do I manage product variables in DITA?

Stanley Doherty, Ph.D.

## Guidepost

**Purpose**: This article provides an overview of DITA variables, how to create them, and how to add them to your DITA topics.

### Learning objective(s):

* Learn what variables are in general and what DITA variables are in particular.
* Learn how to create DITA variables in library topics.
* Learn how to reference these variables in your topics.
* Advanced – Learn how to design variables to work across multiple publications.
* Advanced – Learn how to manage variables using DITA keys.

### Prerequisites:

* Familiarity with creating DITA maps and topics.
* Familiarity with editing DITA markup in a DITA editor.

## Revision history (ACM-014)

| **Date** | **Author** | **Revision summary** |
| --- | --- | --- |
| TBD | Stan Doherty | First public posting |

## Introduction

When writing teams move into DITA, one of the first things that they do is identify the bits of text that will occur repeatedly in their documentation. These include phrases such as product names, company information, release designations, and part numbers. If a team identifies 100 bits of reusable text and estimates that it would reference each of those bits 200 times, it *will* invest time in defining DITA variables for those bits of text.

Learning how to define and use DITA variables for these bits of text is also a great way to gain experience with DITA reuse mechanisms, DITA libraries, and DITA referencing mechanisms.

## Available source code

The DITA source code for examples in this article is available on GitHub.

<https://github.com/acm-sigdoc-structured/dita-variables>

# What are variables in general?

A variable is a name:value pair, for example boston-garden-name=”TD Garden”. The name of the variable (boston-garden-name) does not change, whereas the value (TD Garden) can and often does. In some sense, the name contains the value. When I reference a variable name in DITA, the parser locates the variable definition and returns its current value – whatever it is. Writers can enter the name of the variable in their topics without worrying about having to go back and update a topic if the value of the variable changes. In DITA, every instance of a reference to a named variable will update automatically if the value of the variable changes. If Robert Kraft buys the Boston Garden and changes its name to “Patriot Garden”, every reference to boston-garden-name will resolve to the new value Patriot Garden.

# What are DITA variables and libraries?

A variable in DITA is typically a phrase element that you define once and then reference many times from running text in your topics. Writing teams organize these variables in DITA topics called libraries or warehouses.

Suppose we worked for a small company that manufactured bicycle mirrors. We would certainly collect our company variables in a library topic. Let’s call it library-basic.dita.

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">  
<topic id="library-basic">  
 <title>Basic library of variables</title>  
 <body>  
 <!-- Company information -->  
 <p><keyword id="kw\_company-name">Bob's Bike Mirrors</keyword></p>  
 <p><keyword id="kw\_company-address">5 Farside Dr., Chicago IL</keyword></p>  
 <p><keyword id="kw\_company\_support">888-555-6200</keyword></p>  
 <!-- Product 1 information -->   
 <p><keyword id="kw\_product1-name">Everlast Mirror</keyword></p>  
 <p><keyword id="kw\_product1-partno">2025-01-001</keyword></p>  
 <p><keyword id="kw\_product1-cost">$19.99</keyword></p>  
 <!-- Product 2 information -->  
 <p><keyword id="kw\_product2-name">MaxView Mirror</keyword></p>  
 <p><keyword id="kw\_product2-partno">2025-02-001</keyword></p>  
 <p><keyword id="kw\_product2-cost">$29.99</keyword></p>  
 </body>  
</topic>

Let’s look at one line.

<p><keyword id="kw\_company-name">Bob's Bike Mirrors</keyword></p>

This line defines a variable with the name kw\_company-name and the value Bob's Bike Mirrors. The DITA element containing the variable is <keyword>, which is contained in turn by a paragraph element <p>. The prefix kw\_ for the variable name provides a clue that the value of the variable is contained in a <keyword> element. You can’t insert a <keyword> variable in any context that does not support the <keyword> element.

When you have only a handful of variables, putting them in one library topic that you can review and update makes sense. If you have hundreds of variables, you should organize them in multiple library topics, for example library\_legal.dita, library\_product-names.dita, or library\_support.dita.

# How do I reference DITA variables?

Once you have set up a library topic and defined some variables in it, you can reference those variables from your content topics. You do not need to reference the library topics directly from your DITA maps. When you insert a reference to a variable contained in a library topic, you need to provide enough information in that reference so the DITA parser can find both the library topic and the specific variable in that topic. We’ll work with the most simple type of DITA reference, the content reference (@conref).

Let’s say that I have a topic named advertisement-print.dita, and I want to reference the variable for our company name (kw\_company-name).

<p>  
 Labor Day Sale at  
 <keyword conref="./library-basic.dita#library-basic/kw\_company-name"/>  
</p>

Here’s how the DITA parser interprets and resolves this reference to the variable kw\_company-name:

| <p> | Specifies that the content lives inside a paragraph (<p>) element. |
| --- | --- |
| Labor Day sale at | Provides some running text inside that paragraph. |
| <keyword> | Specifies that the semantic <keyword> element should be used to contain the referenced content. |
| conref | Specifies that the value inside the <keyword> element is referenced from some library topic. We call this a content reference. |
| ./library-basic.dita | Specifies the file path and file name of the library topic. |
| #library-basic | Specifies the topic @id of that library topic. You find this @id at the beginning of every DITA topic, for example <topic id="library-basic">. |
| /kw\_company-name | Specifies the variable name in that library topic. |

When the DITA parser chases down the DITA file name, DITA library topic @id, and <keyword> element @id, it inserts the value of the defined variable at the current location in your DITA topic.



If entering all this markup manually seems cumbersome, it is. DITA editors make the process of locating and inserting variables fairly easy. From the topic into which you want to insert a variable, you specify the location of a library topic and then double-click the name of the variable from the list of variables defined in that library. The editor inserts the complete reference for you.

A screenshot of a computer

Description automatically generated

# Best practices for defining variables

Consider the following best practices when you define variables.

* Organize your variable definitions in one or more library topics.
* Use a filename prefix to identify your library topics. For example, you could use a prefix such as library\_ or lib\_.
* Use a variable name prefix to indicate the element name containing the variable. For example, if the variable is contained in a <ph> element, you might use the prefix ph\_ when you define the variable name ph\_ceo-name.
* Standardize on all upper- or all lower-case characters for your variable names. DITA is case sensitive, so it treats the following variable names as different – ph\_ceo-name and PH\_CEO-NAME.
* Avoid defining separate variables for singular and plural names. Defining the singular variable is easier to maintain and translate. When in doubt, rewrite the sentence to work around the need for a plural variable.
* Add HTML-style comments to clarify what a variable means.

# Single-sourcing content with variables

**NOTE**: This section discusses an advanced application of DITA. If you are mainly interested in learning about the basics of DITA variables, consider skipping this section.

We’ve seen that variables allow writers to change the value of a variable over time so that a new value gets inherited across a DITA doc set. Variables can also support single-sourcing content that is shared across multiple products or publications. For example, if our company, Bob’s Bike Mirrors, sells two products – the Everlast Mirror and the MaxView Mirror – we’d likely publish separate doc sets that share a certain amount of writing. In that shared content, we’ll need to create one variable name, such as kw\_product-name, with the value “Everlast Mirror” when we build the Everlast doc and “MaxView Mirror” when we build the MaxView docs.

One way to have one variable name change its value routinely is to create separate library topics for the Everlast and MaxView and then to swap one or the other into the build process.

The library named library\_everlast.dita would contain the following variable definition:

<p><keyword id="kw\_product-name">Everlast Mirror</keyword></p>

The library named library\_maxview.dita would contain the following variable definition:

<p><keyword id="kw\_product-name">MaxView Mirror</keyword></p>

The content references would point to the variable name kw\_product-name in a library named library\_product.dita. At build time, a build script could copy library\_everlast.dita to library\_product.dita, allowing the variable kw\_product-name to resolve to “Everlast Mirror”. Similarly, copying library\_maxview.dita to library\_product.dita would allow the value of kw\_product-name to resolve to “MaxView Mirror”. Basically, this is using filesystem processes to swap library values. Before DITA 1.2 and the introduction of key-based referencing, this was an all-too-common hack to support the single-sourcing of variables. It works with small-to-medium documentation sets but does not scale up to large doc sets or localization.

# How do I manage variables using DITA keys

**NOTE**: This section discusses an advanced feature in DITA 1.2 and 1.3. If you are mainly interested in learning about the basics of DITA variables, consider skipping this section.

To resolve a value for a variable, each direct reference to that variable requires that it provide a lot of information: library file paths, library filenames, library topic @ids, and variable names.

<keyword conref="./library-basic.dita#library-basic/kw\_company-name"/>

If *anything* changes by way of a file location or name, the direct reference will fail. Teams can’t reorganize the location or contents of their libraries without having to update all the references to those libraries. Again, it works but it is not very scalable.

DITA 1.2 introduced DITA keys, a way to insulate references to a variable from their definition. Unlike direct references using @conref attributes, key-based references point to a key name using the @keyref attribute. The key name and its value are stored in a DITA map. If we had a DITA map named keymap\_maxview.ditamap, we could define a key for the product name.

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">  
<map>  
 <title>Key definition map for the MaxView product. </title>  
 <!-- MaxView product keys -->  
 **<keydef keys="k\_kw\_product-name">  
 <topicmeta>  
 <keywords>  
 <keyword>MaxView Mirror</keyword>   
 </keywords>  
 </topicmeta>  
 </keydef>**  
</map>

A parallel map library could be named keymap\_everlast.ditamap and would contain the following key definition.

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">  
<map>  
 <title>Key definition map for the Everlast product. </title>  
 <!— Everlast product keys -->  
 **<keydef keys="k\_kw\_product-name">  
 <topicmeta>  
 <keywords>  
 <keyword>Everlast Mirror</keyword>   
 </keywords>  
 </topicmeta>  
 </keydef>**  
</map>

In a DITA editor, this is how rootmap\_everlast.ditamap looks.

A screenshot of a computer

AI-generated content may be incorrect.

When I want to build the *MaxView User Guide* from my map rootmap\_maxview.ditmap, I would reference the key definition map keymap\_maxview.ditamap.

<mapref href=”keymap\_maxview.ditamap” processing-role=”resource-only”>

Any reference in any topic to the key name k\_kw\_product-name would resolve to “MaxView Mirror.”

<keyword keyref=”k\_kw\_product-name”/>

Here is how rootmap\_maxview.ditamap looks in a DITA editor.

A screenshot of a computer

AI-generated content may be incorrect.

The attribute @keyref tells the DITA parser to search through all the DITA maps referenced from the root map to find a key definition for k\_kw\_product-name. The parser finds the key name and inserts its value.

To build the *Everlast User Guide* from rootmap\_everlast.ditamap, we would simply reference keymap\_everlast.ditamap instead of keymap\_maxview.ditamap. Any reference to k\_kw\_product-name would resolve to “Everlast Mirror”.

DITA editors make it relatively easy to insert references to keys by listing all the key definitions referenced in key maps referenced from the root map of a publication. Click the key name and the editor inserts the key reference in the topic you are authoring.

A screenshot of a computer

AI-generated content may be incorrect.

Key-based variables allow you to control which key definition libraries are needed for which publications. The same variable name can resolve to many different values depending on which key map you reference from a publication root map. Over time, you can move variable definitions to different key maps, and they will resolve as long as the new key map is referenced from a root map. This allows writing teams to manage the definition of variables over time without ever needing to update references in topics to those variables. It scales nicely.

## Acknowledgements

* TBD - technical review
* TBD - instructional review
* TBD - editorial review

Thank You!

See <https://acm-sigdoc-structured.org> to learn more about committee activities, available resources, and volunteer opportunities.

