Understanding DITA Keys and Key Spaces

Contents

Understanding DITA Keys and Key Spaces

The paper provides commentary and supporting conceptual background for topic 2.1.3.4.3, Key-based addressing, of the DITA 1.2 specification. This paper explains the general purpose and mechanisms behind the DITA key reference facility. It explains the concept of "key spaces" and how they must be constructed by DITA-aware processors. It also explains the rules and algorithms for resolving key references to resources.

The DITA 1.2 key reference facility provides an indirect addressing mechanism that allows authors to create references to things without having to know exactly where or what those things are or will be in the future. Authors create references to keys within topics or maps and those keys are separately bound to specific resources using key definitions within maps. Because the binding of keys to resources is done within maps, the same key can be bound to different resources in different maps.

Processing key references requires determining the effective binding of a given key to a resource in the context of a given root map. Because a root map ultimately defines the set of effective key bindings it is said to establish a *key space*. The processing challenge is in constructing the key space so that it correctly reflects the DITA rules for key definition precedence and also takes applicability (conditional processing) into account appropriately.

Keys and key references

Key references are indirect because the reference as authored is just a key name. The key name is bound to a resource by a separate key definition within a map. In order to resolve the initial reference to the ultimate target resource, the processor must first find the correct key definition and then resolve that to a resource, which then becomes the resource to which the initial key reference resolves.

Indirect addressing is required in order to be able to author topics that contain links to other topics so that they can be used in different map contexts where the links resolve to the appropriate targets in the context of a given root map.

A typical example is a task that applies to many products where the task is generic but it refers to product-specific details. For example, printers often use the same print engine and toner cartridge but have different covers. The task of changing the toner cartridge is the same for all applications of the print engine but the illustrations of the specific printers that use it will be different for each printer model.

If the "change the toner cartridge" task included a reference to a printer-specific illustration you would need a separate copy of the task for each printer model, even though the text of the task is identical for all printers. By contrast, if the task could refer abstractly to the "changing the toner cartridge" illustration and let some other part of the system figure out what specific illustration that should be for a given printer model, then you only need one copy of the task.

The "other part of the system" is the binding of key names to specific resources, which is done using key definition topicrefs within maps. In particular, the root map that is used to produce a particular publication determines the effective binding of keys to resources. Different root maps can define different bindings for the same key.

Key bindings create a *global* key space for a given root map. This means that, within the context of a given root map, all references to a given key name will resolve to the same resource. When a root map includes other maps it forms a *map tree*. Key definitions may occur within any map in the map tree and and there may be multiple definitions for the same key name. When there are multiple definitions for the same key, only one can be the "effective" binding within the scope of a given root map. The rules for determining which definition of a key is the effective one are explained in *Constructing key spaces* on page 10.

For authors, the important (dare I say key) concept is that key bindings are global, not scoped by the map they are defined in. Thus it doesn't matter where a given key reference is relative to where the

effective key binding happens to be in the map tree: the binding is global and therefore all references to the key will resolve to the same resource for a given root map. This behavior is specifically to allow higher maps in the map tree to determine the effective bindings for specific keys. While you might intuitively expect to have key definitions within submap be effective within the scope of that map within a larger map tree, keys don't, and can't, work that way (at least not in DITA 1.2). See more about this subject in the discussion of key space construction.

In the case of our toner cartidge replacement task we know that we need to always have an illustration of the printer that shows where the toner access panel is. We can assign a key name to this illustration that we can then use to indirectly point to the graphic. For example we could assign the key "illustration-toner-cartridge-access-panel" and then use it from an `<image>` element, like so:

```
<task id="changing-toner-cartridge">
  ...
  <context>
    <figure>
      <title>Toner Cartidge Access Panel</title>
      <image keyref="illustration-toner-cartridge-access-panel"/>
    </figure>
  </context>
  ...
</task>
```

Note that here the `<image>` element uses a `@keyref` attribute, rather than an `@href` attribute. The value of the `@keyref` attribute is a *key name*. As the author of this task we only need to know the key name to specify for the graphic, we don't need to know where the graphic is. In fact, because we know that this key will resolve to many different graphics in different printer-specific publications, we know we *can't* know for sure what graphic it will resolve to. Of course, while we're authoring we might have a particular version of this graphic, for example, the illustration for the first printer model this task is used for, or maybe a representative graphic that serves as guidance to illustrators who will create the printer-specific versions. The point is, as the author of this task, we simply don't know or care, ultimately, what graphic the key "illustration-toner-cartridge-access-panel" resolves to as long as it resolves to something in each publication.

The use of the key within the task establishes a requirement for the illustration but it's up to the author of each model-specific publication's map to define the key and bind it to the appropriate illustration.

Within a map, a key definition is created using a `<topicref>` element or any specialization of `<topicref>`. The `<topicref>` element uses the `@keys` attribute to specify the key or keys to be defined and then either points to a resource (a topic, another map, or a non-DITA resource such as a graphic) or uses subelements within `<topicmeta>` to define text to use as the resource (or both).

For our task example, we need to bind the key "illustration-toner-cartridge-access-panel" to the appropriate printer-specific illustration. This means that we must have a separate root map for each different printer model and then within those maps, create the appropriate key definition. For example, for printer model "PR2105" you might have a root map like this:

```
<bookmap>
  <title>Printer PR2105 Operator Manual</title>
  ...
  <topicref
    keys="illustration-toner-cartridge-access-panel"
    href="graphics/PR2105-toner-cartridge-access-panel.svg"
    format="svg"
    processing-role="resource-only"
  />
  ...
  <chapter href="maintenance.dita">
    ...
    <topicref href="common/changing-the-printer-cartridge.dita"/>
    ...
  </chapter>
  ...
</bookmap>
```

The highlighted `<topicref>` element represents a key definition that is binding the key name "illustration-toner-cartridge-access-panel" to the graphic file `graphics/PR2105-toner-cartridge-access-panel.svg`, which is the illustration for the PR2105 model.

Note the `@processing-role` attribute on the `<topicref>` element. This attribute, new in DITA 1.2, indicates whether the topicref contributes to the main navigation tree of the map (the value "normal", which is the default) or serves only as a "resource" that is used by reference from elsewhere ("resource-only"). Because this graphic is not intended to be included in the result publication at this point in the map we must specify a processing role of "resource-only" to tell the processor not to do anything with the graphic other than remember that there is a key associated with it.

The example map also refers to the changing the printer cartridge task topic, representing a specific use of the topic, a topic we know will be used from many different printer-specific publications.

When the task topic is processed *in the context of the PR2105 map* the effective binding of the key name "illustration-toner-cartridge-access-panel" will be to the PR2105 graphic. But when the same topic is used in different maps the binding may be different. For example, the map for the printer model PR3604 would look like this:

```
<bookmap>
  <title>Printer PR3604 Operator Manual</title>
  ...
  <topicref
    keys="illustration-toner-cartridge-access-panel"
    href="graphics/PR3604-toner-cartridge-access-panel.svg"
    format="svg"
    processing-role="resource-only"
  />
  ...
  <chapter href="maintenance.dita">
    ...
    <topicref href="common/changing-the-printer-cartridge.dita"/>
    ...
  </chapter>
  ...
</bookmap>
```

Note that the only difference between the two model-specific maps is the filename of the illustration to use. The key name is the same and the task topic is the same.

Thus the indirection provided by the key reference mechanism allows a single task to be used unchanged in different maps and be associated with the appropriate linked resources.

Note that this indirection separates the concern of information content (what information is required to support a specific topic) from the concern of content configuration management. The author of the topic says, as the content designer, "this topic requires an illustration that shows the toner cartridge access panel" and expresses that general requirement by using a key reference rather than a direct reference to a specific graphic. It is up to the author the map to define the key binding to the appropriate graphic. This makes the content more flexible but does mean there has to be appropriate business processes and communication in place so that the topic author and map author coordinate appropriately (and of course, somebody has to make sure the illustrations get drawn).

The example above uses `<topicref>` elements to define the key bindings in order to make it clear that key definitions are done with topicrefs. Any topic reference can define keys. However, in most cases, you want to define keys as resource-only topicrefs and then use those keys from elsewhere. To make that easier, DITA 1.2 provides the `<topicref>` specialization `<keydef>`, which simply sets the default value for the `@processing-role` attribute to "resource-only". But it can also be useful to associate keys with normal (non resource-only) topicrefs, for example, to enable key-based linking to specific parts of a publication.

For example, in our printer operator guide publications it could make sense to put keys on the chapters and subsection topicrefs too, resulting in a map like this:

```
<bookmap>
  <title>Printer PR3604 Operator Manual</title>
  ...
  <topicref
    keys="illustration-toner-cartridge-access-panel"
    href="graphics/PR3604-toner-cartridge-access-panel.svg"
    format="svg"
    processing-role="resource-only"
  />
  ...
  <chapter
     keys="maintenance-chapter"
```

```
     href="maintenance.dita">
    ...
    <topicref
       keys="changing-printer-cartridge-task"
       href="common/changing-the-printer-cartridge.dita"/>
    ...
  </chapter>
  ...
</bookmap>
```

Here the navigation topicrefs now have keys that allow reference to those topics by key reference. For example, you could have cross references to the changing the printer cartridge task that uses the key rather than a direct href to the topic file:

```
<concept id="indicator-codes">
  <title>Indicator codes and response</title>
  <conbody>
    ...
    <dl>
      ...
      <dlentry>
        <dt>Toner low</dt>
        <dd>Change the toner cartridge.
See <xref keyref="changing-printer-cartridge-task"/>.
        </dd>
      </dlentry>
      ...
    </dl>
    ...
   </conbody>
</concept>
```

Here the author of the topic has indicated a requirement for a changing the printer cartridge task but by using a key doesn't have to worry about the specific task topic. As for the image reference above, the key will be resolved to whatever resource is ultimately bound to the key "changing-printer-cartridge-task" in the context of a specific map. That might be the generic task or, for some printers, it might be a printer-specific task. The author of the indicator codes topic doesn't have to know or care.

Note too that a key reference to a normal topicref establishes a link to a specific use of the target topic, not just to the topic in general. This allows you to have the same topic used multiple times in a map and have unambiguous references to a specific use of that topic, allowing processors to do the right thing.

A topicref that specifies the `@keys` attribute is said to be a "key-defining topicref". A key-defining topicref establishes a binding of a key to a "resource". A resource can be any of the following:

- A topic
- A map
- A non-DITA object (technically, anything that can be addressed by URL that is not itself a DITA document)
- Subelements within the topicref element that provide replacement text for elements like `<term>`, `<keyword>`, and `<xref>`.

A key-defining topicref may both point to a resource and include subelements to use as the link text for links that refer to the key. When a key-defining topicref has subelements but does not point to a resource, you can think of it as a key definition that points to a "null" resource, rather than as a way to define variables.

While you can use key definitions simply to get map-specific text within topics via key references, that use of keys is limited by the fact that keys are global and therefore cannot fully satisfy the general requirements for "variables". As of version 1.2 DITA does not provide a complete mechanism for scoped "variable" text. This is an area that will likely be a focus for DITA 1.3 or future versions.

Note that a key cannot be bound directly to a non-topic element within a topic, it can only be bound to the topic. Key-based references to elements within topics still require specification of the ID of the target element within the topic. This has the side effect that key-based references to specific elements require that the same element ID be used in all topics bound to the key in different maps.

For example, if you needed to create a cross reference from the changing-the-toner-cartridge task to say a table in another topic that must exist for each printer but that will be different for each model, that

table would need to have the same `@id` value in each version of the topic that provides the table. This turns out not to be a problem in practice because usually these types of topics are cloned from a template or an original starting topic, but it does mean that there has to be some coordinate of IDs used in these circumstances because there is no separate indirection mechanism for pointing to elements within topics (at least not in DITA 1.2—there's no technical barrier to such a facility, just a concern about complexity in order to satisfy a relatively rare use case, one for which there is a workaround).

Key-based references to elements within topics use the syntax *`key-name/element-id`*. For example, to create a cross reference to a specific table in another topic you would do this:

```
<task id="changing-toner-cartridge">
  ...
  <context>
    <figure>
      <title>Toner Cartidge Access Panel</title>
      <image keyref="illustration-toner-cartridge-access-panel"/>
    </figure>
    <p>For toner cartridge specifications see
<href keyref="printer-specs-topic/toner-cartridge-specs-table"
   type="table"/>.
  </context>
  ...
</task>
```

The topics that get bound to the key "printer-specs-topic" must each have a `<table>` element with the ID "toner-cartridge-specs-table".

Elements that can make key references

Key references can be used in a number of ways:

- For topic references (`<topicref>`)
- For cross references (`<xref>`, `longquoteref`, etc.)
- For image references (`<image>`)
- For inline elements in order to make those elements into links, including `<ph>`, `<keyword>`, and `<term>`.
- For content references from any element that also allows the `@conref` attribute (most elements except a few like `<title>` that cannot do conref).

Topic reference elements can use key references in place of URI references to point indirectly to topics, maps, or non-DITA resources. A key-defining topicref can itself use a key reference, which means there can be any number of levels of indirection between a key definition and its bound resource. This can be useful for example for imposing your own set of keys onto content that already uses a different set of keys, essentially creating new aliases for the pre-existing keys. It is a basic tenent of computer science that any problem can be solved within another layer of indirection and the key reference facility lets you add as much indirection as your tools and authors will tolerate.

When elements, such as `<topicref>` and `<xref>` allow both `@href` and `@keyref`, when both are specified the `@href` is used only if the key reference cannot be resolved, either because the key is not defined or because the bound resource cannot be retrieved for whatever reason.

Elements that allow `@keyref` but not `@href`, such as `<ph>`, `<keyword>`, and `<term>`, become links when they specify `@keyref` and the key is bound to a resource. If the element's content is empty then the "link text" (that is, the text rendered for the element) is taken from the nearest key-defining topicref that specifies a `<linktext>` or `<keyword>` element as a descendant of itself. This mechanism is intended primarily to make it easy to author links from mentions of things to topics that define them or otherwise relate to them so that those links will be appropriate in different map contexts.

For example, first mentions of technical terms are often linked to their glossary definitions. You can do this using `@keyref` from `<term>` or `<keyword>` rather than a separate cross reference. In addition, the key definition can provide the actual text of the term so that you don't have to change the reference if the term changes.

For example, in the printer example it would make sense to have the printer model number defined in the map as we know that each publication will be specific to a single printer model. Within the task you would have something like this:

```
<task id="changing-toner-cartridge">
  ...
  <context>
    <figure>
      <title><keyword keyref="printer-model-number"/>
Toner Cartidge Access Panel</title>
      <image keyref="illustration-toner-cartridge-access-panel"/>
    </figure>
    <p>For toner cartridge specifications see
<href keyref="printer-specs-topic/toner-cartridge-specs-table"
   type="table"/>.
  </context>
  ...
</task>
```

This example adds a <keyword> element with a @keyref attribute to the key name "printer-model-number". Note that the <keyword> element is empty.

Within the printer-specific maps we add a definition for the key name "printer-model-number":

```
<bookmap>
  <title>Printer PR3604 Operator Manual</title>
  ...
  <keydef keys="printer-model-number">
    <topicmeta>
      <linktext>PR3604</linktext>
    </topicmeta>
  </keydef>
  <topicref
    keys="illustration-toner-cartridge-access-panel"
    href="graphics/PR3604-toner-cartridge-access-panel.svg"
    format="svg"
    processing-role="resource-only"
  />
  ...
  <chapter
     keys="maintenance-chapter"
     href="maintenance.dita">
    ...
    <topicref
       keys="changing-printer-cartridge-task"
       href="common/changing-the-printer-cartridge.dita"/>
    ...
  </chapter>
  ...
</bookmap>
```

When the task topic is processed the <keyword> element will be rendered as though the text of the <linktext> element in the key definition had been its content. Because the key definition does not also point to a resource the <keyword> element will not be a link (because there's nothing to link to, e.g., the keydef has a null resource).

As for navigation links, content references can use key references in place of, or in addition to, URI references. This allows you to have re-useable content references without having to know in advance what topic will provide the specific element referenced.

For example, say the changing the toner cartridge task is generic except for one step which must be model specific for some reason. As for the illustration and the model number, you can use a conkeyref to establish the requirement for a model-specific step and let each model-specific map hook up the appropriate topic.

The conkeyref in the task topic would look like this:

```
<task id="changing-toner-cartridge">
  ...
  <context>
    <figure>
      <title><keyword keyref="printer-model-number"/>
Toner Cartidge Access Panel</title>
      <image keyref="illustration-toner-cartridge-access-panel"/>
    </figure>
    <p>For toner cartridge specifications see
<href keyref="printer-specs-topic/toner-cartridge-specs-table"
   type="table"/>.
```

```
  </context>
  <steps>
    <step conkeyref="model-specific-steps/changing-toner-cartridge-step-one"/>
    <step>
      <cmd>...</cmd>
    </step>
    ...
  </steps>
  ...
</task>
```

Here the `<step>` element uses the `@conkeyref` attribute rather than `@conref`. The semantic is the same (content use by reference) but the pointer is indirect rather than direct.

The topic that provides the step would be a "resource" topic, intended only as a container of steps and other elements to be used by reference. It could look something like this:

```
<task id="reusable-steps">
  <title>Task Component Resource Topic for Printer Model PR3604</title>
  <taskbody>
    <steps>
      <step id="changing-toner-cartridge-step-one">
        <cmd>Lift the handle...</cmd>
      </step>
      ...
    </steps>
  </taskbody>
</task>
```

Each model-specific version of this topic would need to use the `@id` value "changing-toner-cartridge-step-one" for its version of that step.

Within each product-specific map you would have a key definition that binds the key name "model-specific-steps" to the appropriate resource topic:

```
<bookmap>
  <title>Printer PR3604 Operator Manual</title>
  ...
  <keydef keys="printer-model-number">
    <topicmeta>
      <linktext>PR3604</linktext>
    </topicmeta>
  </keydef>
  <keydef keys="model-specific-steps"
          href="resources/PR3604/PR3604-reusable-steps.dita"
  />
  <keydef
    keys="illustration-toner-cartridge-access-panel"
    href="graphics/PR3604-toner-cartridge-access-panel.svg"
    format="svg"
  />
  ...
  <chapter
     keys="maintenance-chapter"
     href="maintenance.dita">
    ...
    <topicref
       keys="changing-printer-cartridge-task"
       href="common/changing-the-printer-cartridge.dita"/>
    ...
  </chapter>
  ...
</bookmap>
```

Note that this example uses `<keydef>` rather than `<topicref>` as `<keydef>` sets the `@processing-role` attribute to "resource-only" by default, which is what we want for our reusable steps topic, since it is literally a resource used from other topics.

Again, because keys are global you can only use this technique for root maps that reflect a single model. You cannot have two different models in the same publication and use a single key to get different results in different parts of the publication, at least not in DITA 1.2.

Constructing key spaces

In the simple case a given root map does not include any submaps and every key name is defined by exactly one key-defining topicref. However, this is neither the only case nor the typical case.

Most non-trivial publications involve maps and submaps and it will often be the case that the same key name is defined by multiple key-defining topicrefs within the context of a given root map and its submaps.

The processing challenge is to determine which of multiple definitions of a given key is the correct one.

The general design intent of the DITA 1.2 key reference facility is that a given root map establishes a *global* key space within which a given key name has at most one binding and where earlier and "higher" definitions take precedence over later and "lower" definitions.

By "higher" and "lower" is meant the position of maps within the tree of maps descending from the root map. Maps that are closer to the root map within the map tree take precedence over maps that are lower in the map tree. Maps that are referenced earlier within a given using map take precedence over maps referenced later.

This design allows maps that use other maps by reference to override the key definitions in the referenced maps.

If the key definition facility were not defined in this way it would be impossible for higher-level maps to control the effective key values.

Note that because the key space defined by a given root map is global, it means that you cannot have different effective bindings for a given key active at different places within a map tree. For example, you cannot use keys to allow the same key for different language versions of term within a single publication produced from a single root map. This is a limitation to be sure, but one that was necessary to accept in DITA 1.2 in order to get any sort of indirection facility defined in a timely fashion. It is likely that future versions of DITA will extend the key definition facility in order to provide other key scopes. At the same time, it is likely that additional "variable" facilities will be defined that will serve to satisfy requirements for locally-scoped, dynamically-resolved values that the key reference facility cannot today (and may never) satisfy. In addition, there are processing techniques that allow you to get the effect of locally-scoped keys without changing the base key processing semantics. Many of these have been discussed on the Yahoo DITA Users group.[1]

The formal definition of key space construction says that within a given map document the first key definition in document order is the effective definition. Within the map tree, the first definition within a *breadth-first* traversal of the map tree takes precedence.

"Document order" means the order the start tags of each element in the XML document tree are encountered, technically a *depth-first* traveral of the element tree. A depth-first traversal goes from parent to child until a leaf node is reached, and only then goes to the next sibling of that leaf (or the next sibling of its nearest ancestor with a sibling) and so on down the tree.

A *bread first* traveral goes from one node to its siblings before then going to each of the children's children. See *Figure 1: Map Tree of Four Maps* on page 12.

Because the rule is "first in map document in document order then first in map tree in breadth-first order", when the map tree consists of exactly one map, the first definition in document order wins. When the map tree consists of two or more maps, the first definition within the map nearest the root map in breadth-first order takes precedence, with the root map always being first.

Consider a system of four maps: a root (Root Map 1), two submaps, Submap 1 and Submap 2, both referenced from the root map in the order Submap 1 followed by Submap 2, and submap Submap 3, referenced from Submap 1.

---

[1] If you have any question about the necessity of a global key space, consider the case of a topic referenced from one submap that references a key that is defined in both of two other submaps but not defined in the root map. If the key space was not global it would be ambiguous in that case which of the two possible key binding to use and there would be no way for authors to indicate which they meant. Likewise, if keys were scoped by submaps, it would be impossible for topics within the scope of a given submap to refer to a key defined in both its map scope and a different map scope. Thus, without a way of establishing as part of a key reference the scope (key space) context, the only option is to have a global key space so that all key references are unambiguous.

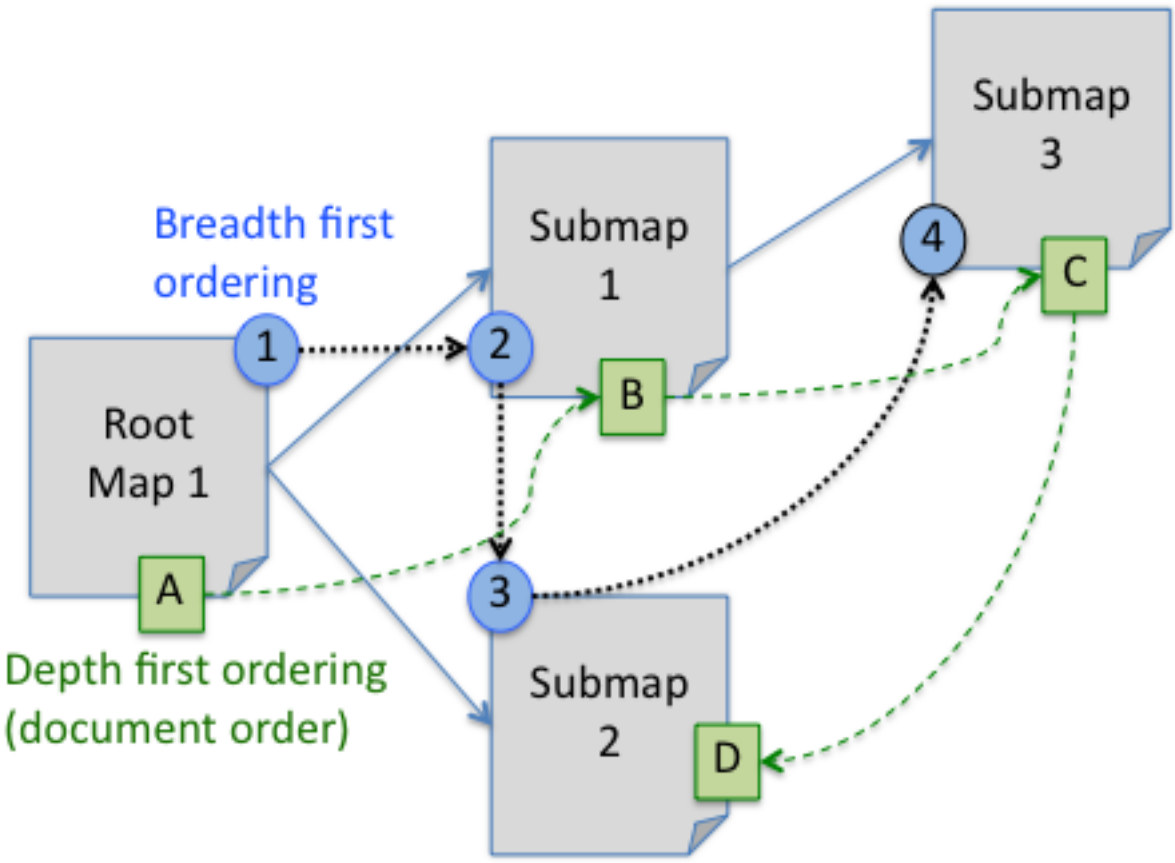

Figure 1: Map Tree of Four Maps

Any key defined in the root map will take precedence over the same key defined in either of the submaps. A key defined in Submap 1 will take precedence over the same key defined in Submap 2 because Submap 1 occurs before Submap 2 in the map tree. Finally any key defined in Submap 2 will take precedence over the same key defined in Submap 3 because Submap 2 occurs before Submap 3 in a breadth-first traversal of the map tree.

Note that the position of key references relative to key definitions is not important: the key space is global and must be determined before any key references can be resolved.

Consider this set of key definitions in the root map:

```
<map>
  <title>Root Map 1</title>
  <topicgroup>
     <keydef keys="key-01" href="topic-1A.dita"/>
     <keydef keys="key-01" href="topic-1B.dita"/>
  </topicgroup>
  <keydef keys="key-01" href="topic-1C.dita"/>
</map>
```

Here there are three key-defining topicrefs for the key name "key-01". The effective definition is the binding to topic-1A.dita, because that is the first definition in document order (depth-first traversal of the element tree within the map document). The breath-first rule only applies to the tree of maps, not to the tree of elements within a document.

Now consider the root map with a reference to Submap 1 added:

```
<map>
  <title>Root Map 1</title>
  <topicgroup>
```

```
    <mapref href="submap-01.ditamap"/>
  </topicgroup>
  <topicgroup>
     <keydef keys="key-01" href="topic-1A.dita"/>
     <keydef keys="key-01" href="topic-1B.dita"/>
  </topicgroup>
  <keydef keys="key-01" href="topic-1C.dita"/>
</map>
```

If Submap 1 also defines a binding for key name "key-01" it cannot be effective because, by the breadth-first map tree traversal rule, any definition in the root map takes precedence over any definitions in submaps, even though the reference to the submap happens to occur before any of the key definitions in the root map.

If we now add the reference to Submap 2 we establish the order of those maps in the map tree:

```
<map>
  <title>Root Map 1</title>
  <topicgroup>
    <mapref href="submap-01.ditamap"/>
  </topicgroup>
  <mapref href="submap-02.ditamap"/>
  <topicgroup>
     <keydef keys="key-01" href="topic-1A.dita"/>
     <keydef keys="key-01" href="topic-1B.dita"/>
  </topicgroup>
  <keydef keys="key-01" href="topic-1C.dita"/>
</map>
```

Again, it is the document (depth-first) order of the references to the maps that determines their ordering in the map tree. Thus, even though in this example the reference to Submap 2 comes first in a *breadth*-first traversal of the elements within the root map, that is not relevant because it is the document order that is important. Thus this markup produces the map tree shown in *Figure 1: Map Tree of Four Maps* on page 12.

To complete the map tree Submap 1 must include Submap 3:

```
<map>
  <title>Submap 1</title>
  <mapref href="submap-03.ditamap"/>
  <keydef keys="key-01" href="topic-1D.dita"/>
</map>
```

And let Submap 3 look like this:

```
<map>
  <title>Submap 3</title>
  <keydef keys="key-01" href="topic-1E.dita"/>
  <keydef keys="key-02" href="topic-2B.dita"/>
</map>
```

Finally, Submap 2 looks like this:

```
<map>
  <title>Submap 2</title>
  <keydef keys="key-01" href="topic-1F.dita"/>
  <keydef keys="key-02" href="topic-2A.dita"/>
</map>
```

The effective value of key "key-01" is set in the root map because the root map has at least one definition for that key.

The effective value of the key "key-02" is determined by its first position within the map tree because it is not defined in the root map.

In this case, key-02 is defined in two maps: Submap 2 and Submap 3. The effective definition is the definition in Submap 2 (topic-2A.dita) because Submap 2 comes first in the map tree in a breadth-first traversal.

The key space defined by this map tree can be represented by a table that has one row for each unique key and indicates at least the effective definition, as shown in *Table 1: Key space table for Root Map 1* on page 14.

Table 1: Key space table for Root Map 1

Key Name	Key Definitions (In precedence order)	Containing Map
key-01	**<keydef keys="key-01" href="topic-1A.dita"/>**	rootmap.ditamap
	<keydef keys="key-01" href="topic-1B.dita"/>	rootmap.ditamap
	<keydef keys="key-01" href="topic-1C.dita"/>	rootmap.ditamap
	<keydef keys="key-01" href="topic-1D.dita"/>	submap-01.ditamap
	<keydef keys="key-01" href="topic-1F.dita"/>	submap-02.ditamap
	<keydef keys="key-01" href="topic-1E.dita"/>	submap-03.ditamap
key-02	**<keydef keys="key-02" href="topic-2A.dita"/>**	submap-02.ditamap
	<keydef keys="key-02" href="topic-2B.dita"/>	submap-03.ditamap

In this table, the key definitions for a given key are listed in precedence order, first by document order within a single map and then by breadth-first order within the map tree. The first key definition for each key is therefore the effective one, as shown by the highlighting.

Note that the root map serves as the identifier of the key space because it is the root map that ultimately determines the effective bindings of all the keys. Different root maps represent distinct key spaces.

By the same token, you cannot resolve any key until you have established the active key space for the resolution attempt. This has important implications for authoring and component management, about which more later.

To more clearly demonstrate the fact that different root maps establish different key spaces, let us now create Root Map 2, which omits the definitions of key-01 in the root and includes Submap 1 and Submap 2, but in the reverse order from Root Map 1:

```
<map>
  <title>Root Map 2</title>
  <topicgroup>
    <mapref href="submap-02.ditamap"/>
    <mapref href="submap-01.ditamap"/>
  </topicgroup>
</map>
```

Now Submap 2 comes before Submap 1 in the map tree and that makes the effective value of key-01 the binding defined in Submap 2, namely "topic-1F.ditamap". The key space table for key space Root Map 2 looks like this:

Table 2: Key space table for Root Map 2

Key Name	Key Definitions (In precedence order)	Containing Map
key-01	<keydef keys="key-01" href="topic-1F.dita"/>	submap-02.ditamap
	<keydef keys="key-01" href="topic-1E.dita"/>	submap-03.ditamap
key-02	**<keydef keys="key-02" href="topic-2A.dita"/>**	submap-02.ditamap
	<keydef keys="key-02" href="topic-2B.dita"/>	submap-03.ditamap

Any topic that has a link to key "key-01" will link to different target topics when processed in the context of Root Map 1 than when processed in the context of Root Map 2.

You might wonder why the key space table shows all the bindings even though only one can be effective within a given key space. There are several reasons:

- For the purposes of this example it makes it clear what the precedence order of the different definitions is.

- From an analysis and debugging standpoint it can be useful to know what all the definitions of a given key are in a given keyspace. Thus key space constructing processors should expect to build this sort of table so that they have all the information about all the key definitions and can therefore report it back to users when requested.
- In a CMS system it may be useful to maintain repository-wide knowledge of all the key definitions in all the maps, with determination of effective key spaces done more or less dynamically. Some sort of "key database" where the key is the primary lookup key would look very much like this key definition table, at least at its core.
- When applicability (conditional processing) is in play, it may be necessary to hold all potentially effective bindings for a key and determine the effective binding in the context of a specific set of conditions. This is especially important in authoring support systems where authors may want to see a in-editor view of the data as determined by a specific condition set dynamically applied. Unless the key space table includes at least all potentially-applicable key definitions, this type of dynamic view cannot be provided.
- At some point in the future it is likely that there will be a way to define the resolution scope for specific key references, which means that any definition of a key could potentially be effective based on specific processing-time or resolution-time scope settings.

Finally, note that the map tree is constructed using only directly-addressed maps. This is because you can't resolve any key-based map references until you construct the key space and you can't construct the key space until you have constructed the map tree.

This implies that there are conceptually at least two passes in the processing of the map: a first pass to construct the map tree and key space and a second pass to construct the full map tree, taking any indirect map references into account. The output of the second step would then be the input to any additional processing applied to the map.

Key definitions and applicability (conditional processing)

When all key definitions are unconditional, constructing the key space is relatively easy. However, when key definitions or map references are conditional, things can get complicated very quickly.

The issue is that the DITA specification allows processors to choose when conditional processing is applied: before or after key space construction. In addition, some processors may need to apply conditional processing as late as possible, such as editors and component management systems that need to able to provide different views of the source documents reflecting different sets of active conditions.

Note also that only the filtering aspect of conditional processing is relevant here—flagging is a rendition issue and does not affect key space construction in any way.

When filtering is applied before key space construction then the key space is fixed, meaning that any excluded key definitions are eliminated before key space construction and therefore cannot be represented in the final key space.

When filtering is applied after key space construction then the effective value for a given key cannot be completely determined except in in the context of a specific set of active conditions.

For example, say that the definitions of key-01 in the Root Map 1 example actually reflect the bindings appropriate for different conditions, resulting in this markup:

```
<map>
  <title>Root Map 1</title>
  <topicgroup>
    <mapref href="submap-01.ditamap"/>
  </topicgroup>
  <topicgroup>
     <keydef
        platform="platform-A"
        keys="key-01"
        href="topic-1A.dita"/>
     <keydef
        platform="platform-B"
        keys="key-01"
```

```
        href="topic-1B.dita"/>
    <keydef
       platform="platform-C"
       keys="key-01"
       href="topic-1C.dita"/>
  </topicgroup>
</map>
```

Now each of the definitions of key "key-01" has a different value for the `@platform` property. The effective binding will depend on the setting for the platform property either at the time the key space is constructed (filtering applied first) or at the time the key reference is resolved (filtering applied after key space construction).

In serial batch processors like the Open Toolkit or DITA2Go, the effect is the same either way for a given active property set. But for interactive systems like editors and component management systems the difference is important.

To make the example a little more interesting, lets add an unconditional definition for the key as well as a duplicate conditional definition:

```
<map>
  <title>Root Map 1</title>
  <topicgroup>
    <mapref href="submap-01.ditamap"/>
  </topicgroup>
  <topicgroup>
     <keydef
        platform="platform-A"
        keys="key-01"
        href="topic-1A.dita"/>
     <keydef
        platform="platform-A"
        keys="key-01"
        href="topic-1H.dita"/>
     <keydef
        platform="platform-B"
        keys="key-01"
        href="topic-1B.dita"/>
     <keydef
        platform="platform-C"
        keys="key-01"
        href="topic-1C.dita"/>
     <keydef
        keys="key-01"
        href="topic-1G.dita"/>
  </topicgroup>
</map>
```

Now there are four possible effective bindings, any one of which could be the effective one depending on the active value for the platform property. The binding to topic "topic-1H.dita" cannot ever be effective because it comes after a definition for the same key name with the same set of conditions (platform = "platform-A"). But the remaining four definitions could each be active under different sets of conditions.

Thus, if a processor needs to provide resolution-time determination of key bindings based on conditions, it needs to add a "conditions" column to the key space table, as shown in *Table 3: Key space table for Root Map 1 with applicability conditions* on page 16.

Table 3: Key space table for Root Map 1 with applicability conditions

Key Name	Key Definitions (In precedence order)	Containing Map	Applicable Conditions
key-01	**<keydef keys="key-01" href="topic-1A.dita"/>**	rootmap.ditamap	platform = platform-A
	<keydef keys="key-01" href="topic-1B.dita"/>	rootmap.ditamap	platform = platform-B
	<keydef keys="key-01" href="topic-1C.dita"/>	rootmap.ditamap	platform = platform-C

Key Name	Key Definitions (In precedence order)	Containing Map	Applicable Conditions
	<keydef keys="key-01" href="topic-1D.dita"/>	rootmap.ditamap	None
	<keydef keys="key-01" href="topic-1D.dita"/>	submap-01.ditamap	None
	<keydef keys="key-01" href="topic-1F.dita"/>	submap-02.ditamap	None
	<keydef keys="key-01" href="topic-1E.dita"/>	submap-03.ditamap	None
key-02	**<keydef keys="key-02" href="topic-2A.dita"/>**	submap-02.ditamap	None
	<keydef keys="key-02" href="topic-2B.dita"/>	submap-03.ditamap	None

Now a processor resolving "key-01" sees that all four of the definitions in the root map are potentially effective and it must therefore ask what value for the platform condition to use when resolving the key reference. Depending on the value specified the result will be one of the four possible values.

The implication for dynamic resolution processors such as editors and component management systems are:

1. Processors must inspect the set of conditions on each key definition in order to determine the set of potentially applicable key definitions. For a given key, the first definition with a unique set of conditions is potentially effective, second and subsequent definitions with the same condition set cannot be potentially effective (by the precedence rules for key definitions).
2. Processors must maintain knowledge of the applicable conditions for each potentially-effective key definition.
3. Resolution requests must allow the specification of active conditions as one of the parameters to the resolution request. That is, given an API method like "resolveKeyReference()", the parameters must include not just the key space (or identifier of the root map that establishes the key space) but a set of active conditions. Likewise, user interfaces that allow users to see the effect of resolved references must provide a way to specify active conditions. This could be either through direct specification or through prior configuration of the active conditions.

Key definitions and content reference resolution

Because content references may use key references, conref resolution processing cannot be fully performed until the key space has been constructed. However, processors may choose to do resolution of direct conrefs before doing key space construction. This could result in different effective key spaces in some cases, although it should be rare for key definitions to be managed in maps through conref rather than through topicref-based inclusions.