Eberlein Consulting

DITA training, information architecture, and strategy

# Key-based addressing

# Contents

# Indirect key-based addressing

DITA allows references to be made indirectly by referencing a name for a link target, instead of referencing the location of that target. This name is called a key. The DITA key-reference mechanism provides a layer of abstraction so that resource locations can be defined globally at the DITA map level instead of locally in each topic.

When using DITA topics in the context of different maps, it is often necessary to have relationships resolve to different resources. For example, a content reference to a `<keyword>` element that contains a product name might need to resolve to a different `<keyword>` element when used in a different product-specific map. The DITA key-reference mechanism provides an indirect addressing mechanism that separates references (topicrefs, conrefs, cross references, etc.) from the direct address of the target. (A direct address is the address specified on the element that references the key, for example, through the `@href` or `@conref` attribute.) Linking elements can refer to key names; the key names then are bound to specific resources by maps. Different maps can bind the same key names to different resources. The binding of key names to resources is determined at processing time based on the current set of key definitions for the map context, rather than from a static binding that is created when a topic or map is authored.

## Core concepts for working with keys

The concepts described below are critical for a full understanding of keys and key processing.

### Core definitions related to keys

**key**

A name for a resource.

**key definition**

A `<topicref>` element or specialization that binds one or more key names to resources.

**key reference**

An attribute that references a key, such as `@keyref` or `@conkeyref`.

**Comment by chris.nitchie**
Look for places where we talk about elements as key references to make sure we're being consistent.

**key space**

A list of key definitions that are used to resolve key references.

**effective key definition**

The definition for a key within a key space that is used to resolve references to that key. A key might have multiple definitions within a key space, but only one of those definitions is effective.

**key scope**

A map or section of a map that defines its own key space and serves as the resolution context for its key references.

### Key definitions

A key definition binds the key to a number of resources:

- Any content referenced directly by the `@href` attribute or indirectly by the `@keyref` attribute. References to the key definition are considered references to this content.
- The contents of any `<topicmeta>` child element. Those contents can be used to populate the content of elements that reference the definition.

If a key definition does not contain a `<topicmeta>` element, and does not reference a resource by `@href` or `@keyref`, it is nonetheless a valid key definition. References to the key definition are considered resolvable, but no linking or content population occurs.

## Key scopes

All key definitions and key references exist within a key scope. Within a map hierarchy, key scopes are bounded by

- The root map.
- The root element of submaps whose root elements specify the `@keyscope` attribute.
- Any `<topicref>` elements or `<topicref>` specializations that specify the `@keyscope` attribute.

The `@keyscope` attribute declares a name or names for the scope. If `@keyscope` is specified both on the root element in a map and on a `<topicref>` element that references that map, only one scope is introduced; its names are the union of the names that are specified by both `@keyscope` attributes.

## Key spaces

**Comment by Kristen Eberlein**

I hate the following phrases:

- key scope's key space
- parent scope's key space

A key scope is associated with exactly one key space. That key space contains all key definitions that are located directly within the scope; it may also contain definitions that exist in other scopes. Specifically, a key scope's key space is comprised of the following key definitions, in order of precedence:

1. All key definitions from the parent key scope's key space, if any.
2. Key definitions within the scope-defining element, including those defined in directly-addressed, locally-scoped submaps, but excluding those defined in child scopes.
3. The key definitions from child scopes, with each key prepended by the child scope name followed by a period. If a child scope has multiple names, the keys in that scope are addressable from the parent scope using any of the scope names as a prefix.

**Note:** Because of rules 1 and 3, the key space associated with a child scope includes the scope-qualified copies of its own keys that are inherited from the parent scope's key space, as well as those from other "sibling" scopes.

The key space associated with a key scope is used to resolve all key references that occur immediately within that scope. Key references in child scopes are resolved using the key spaces that are associated with those child scopes.

## Effective key definitions

A key space can contain many definitions for a given key, but only one definition is effective for the purpose of resolving key references.

When a key has a definition in a parent scope's key space, that definition is ceffective. Otherwise, a key definition is effective if it is first in a breadth-first traversal of the locally-scoped submaps beneath the scope-defining element. Put another way, a key definition is effective if it is the first definition in the shallowest map for that key. This allows higher-level map authors to override keys defined in referenced submaps.

**Note:** A key definition that specifies more than one key name in its `@keys` attribute may be the effective definition for some of its keys but not for others.

Within a key scope, keys do not have to be defined before they are referenced. The key space is effective for the entire scope, so the order of key definitions and key references relative to one another is not significant. This means that all key spaces for a root map must be determined before any key reference processing can be performed, and that maps referenced solely by key reference have no bearing on key space contents.

For purposes of key definition precedence, the scope-qualified key definitions from a child scope are considered to occur at the location of the scope-defining element within the parent scope.

**Comment by chris.nitchie on 7/25/14**
This is controversial and needs more thought.

```
map
   sub-map scope="scopeA"
      sub-map scope="scopeB"
         keydef="MYKEY"
   sub-map scope="scopeA.scopeB"
     keydef MYKEY

key scopeA.scopeB.MYKEY=?
```

Per traditional rules, the snwer is the second definition; per this sentence, it's the first, violating existing stuff.

# Using keys to address DITA elements

For topic references, image references, and navigation link relationships (`<link>`, `<xref>`, and elements that take the `@keyref` but not the `@href` attribute), resources can be addressed by key using the `@keyref` attribute. For content reference relationships, resources can be addressed by key using the `@conkeyref` attribute.

## Syntax

For references to topics, maps, and non-DITA resources, the value of the `@keyref` attribute is simply a key name (for example, `keyref="topic-key"`).

For references to non-topic elements within topics and non-topicref elements within maps, the value of the `@keyref` attribute is a key name, a slash ("/"), and the ID of the target element: `keyref="topic-key/some-element-id"`.

If both `@keyref` and `@href` attributes are specified on an element, the `@href` value *MUST* be used as a fallback address when the key name is undefined. If both `@conkeyref` and `@conref` attributes are specified on an element, the `@conref` value *MUST* be used as a fallback address when the key name is undefined.

**Comment by robander on September 16, 2014**
After discussing with Chris Nitchie, I've removed 2 copies of a clause from the previous paragraph, highlighted below in the original version:

If both `@keyref` and `@href` attributes are specified on an element, the `@href` value *MUST* be used as a fallback address when the key name is undefined, ***and SHOULD be used as a fallback address when the key name is defined but the key reference cannot be resolved to a resource***. If both `@conkeyref` and `@conref` attributes are specified on an element, the `@conref` value *MUST* be used as a fallback address when the key name is undefined, ***and SHOULD be used as a fallback address when the key name is defined but the key reference cannot be resolved to a resource***.

In our reading, the portion about a key being defined but not resolved conflicts with other language about processing, which for example states that key definitions without a link target or with linking="none" result in no

link. That language is clear and conflicts with the removed language here, which suggested that given a valid key definition with no link target, processors should use the local @href as fallback.

The language could also have been interpreted as "If the resource referenced by a key definition does not exist, @href SHOULD be used as fallback", which does not seem like something we can or should enforce. If the key definition points to example.com, but that server is down, an application would seemingly have to 1) know this, 2) assume it does not exist, and 3) switch to using @href.

### Example

For example, consider this topic in the document `file.dita`:

```
<topic id="topicid">
 <title>Example referenced topic</title>
 <body>
  <p id="para-01">Some content.</p>
 </body>
</topic>
```

and this key definition:

```
<map>
  <topicref keys="myexample"
    href="file.dita"
  />
</map>
```

A keyref of the form "`myexample/para-01` resolves to the `<p>` element in the topic. The key reference would be equivalent, in the context of this map, to the URI reference `file.dita#topicid/para-01`.

A key reference to a `<topicref>` element where the linking element specifies a format value of "ditamap" addresses the `<topicref>` element itself as though the `<topicref>` element had been addressed by ID. In particular, a `<topicref>` with a key reference to another `<topicref>` and a format value of "ditamap" is a use of the map branch rooted at the referenced `<topicref>`.

## Processing key references

Key references can resolve as links, as text, or as both. Within a map, they can also be used to create or supplement information on a topic reference. This topic covers information that is common to all key processing, regardless of how the key is used.

### Keys and conditional processing

The effective key definitions for a key space might be affected by conditional processing (filtering). Processors must perform conditional processing before determining the effective definition for a key reference. However, processors might determine key space contents before filtering. Consequently, different processors might produce different effective bindings for the same map when there are key definitions that might be filtered out based on their filtering attributes.

### Reusing a topic in multiple key scopes

If a topic that contains key references is reused in multiple key scopes within a given root map such that its references resolve differently at each use context, processors *MUST* produce multiple copies of the source topic in resolved output for each distinct set of effective key definitions that are referenced by the topic. In such cases, authors can use the `@copy-to` attribute to control output generation.

### Keys in peer root maps

For references to keys in peer root maps, if the peer map is unavailable for processing or the key reference cannot otherwise be resolved, the key reference is processed as though there was no key reference. If the peer map is available, the key reference is processed normally.

### Error conditions

It is an error for an empty element to have a key reference with an undefined key and have no `@href` attribute for fallback. In this case, an implementation *MAY* give an error message, and *MAY* recover from this error condition by leaving the key reference element empty.

## Processing key references on `<topicref>` elements

While `<topicref>` elements are used to define keys, they also can reference keys defined elsewhere. This topic explains how to evaluate key references on `<topicref>` and its specializations.

For topic references that use the `@keyref` attribute, the effective value of the `<topicref>` element is determined in the following way:

**Comment by Kristen Eberlein**
Might want to change the <ul> to <dl>, with appropriate headers.

- The effective resource bound to the `<topicref>` element is determined by resolving all intermediate key references. Each key reference is resolved either to a resource addressed directly by URI reference in an `@href` attribute, or to no resource. Processors *MAY* impose reasonable limits on the number of intermediate key references that they will resolve. Processors *SHOULD* support at least three levels of key references.

  **Note:** This rule applies to all topic references, including those that define keys. The effective bound resource for a key definition that uses the `@keyref` attribute cannot be determined until the key space has been constructed.

  The attributes that are common to a key definition element and a key reference element using that key, other than the `@keys`, `@processing-role`, and `@id` attributes, are combined as for content references, including the special processing for the `@xml:lang`, `@dir`, and `@translate` attributes. There is no special processing associated with either the `@locktitle` or the `@lockmeta` attributes when attributes are combined.
- Content from a key reference element and a key-defining element is combined following the rules for combining metadata between maps and other maps and between maps and topics. The `@lockmeta` attribute is honored when metadata content is combined.
- The combined attributes and content cascade from one map to another or from a map to a topic, but this is controlled by existing rules for cascading, which are not affected by the use of key references.

If, in addition to the `@keys` attribute, a key definition specifies a `@keyref` attribute that can be resolved after the key resolution context for the key definition has been determined, the resources bound to the referenced key definition take precedence.

**Comment by robander**
NEEDS EXAMPLE OF THIS PARAGRAPH

## Processing key references to create or redirect links

Keys can be used to create or redirect links and cross references. This topic explains how to evaluate key references on links and cross references to determine a link target.

When a key definition is bound to a resource that is addressed by `@href` or `@keyref` and does not specify "none" for the `@linking` attribute, all references to that key definition become navigation links to the bound resource. When a key definition is not bound to a resource or specifies "none" for the `@linking` attribute, references to that key do not become navigation links.

When a key definition has no `@href` value and no `@keyref` value, references to that key will not result in a link, even if they do contain an `@href` attribute of their own. If the key definition also does not contain a `<topicmeta>` subelement, empty elements that refer to the key (such as `<link keyref="a"/>` or `<xref keyref="a" href="fallback.dita"/>`) are removed.

If a referencing element specifies a key reference to an undefined key, it is processed as if there were no key reference, and the value of the `@href` attribute is used as the reference. If the `@href` attribute is not specified either, the element is not treated as a navigation link.

# Processing key references to generate text or link text

Key references can be used to pull text from the key definition. This topic explains how to generate text from a key definition, regardless of whetherthe key reference also results in a link.

**Note:** The processing described in this topic is unrelated to the `@conkeyref` attribute. In that case `@conkeyref` is used to determine the target of a `@conref` attribute, after which the normal `@conref` rules apply.

Empty elements that include a key reference with a defined key might get their effective content from the key definition. Empty elements are defined as elements which meet the following criteria:

- Have no text content
- Have no sub-elements

  **Comment by robander on September 16 2014**
  There is a recognized edge case that could confuse processors. What if my specialization has a required kid. For validity, I must include that kid. (Same issue tends to come up with conref on something like <step> or <table>). For example, if I have an image with required <alt> child, to be valid it must be: <myImage keyref="a"><alt/></myImage>. This can never be empty, thus can never pull the alt-text along with @href.

  Not sure what the alternative is though. Originally considered text: "Have no sub-elements that contain text or result in text". But as Chris pointed out this is troublesome - how can a processor know if a child element will result in text? Will <myData/> result in text? What if it results in text for PDF but not HTML - then does my effective content get pulled from the key for one and not the other? What if you have <keyword> child with @conref, but @conref doesn't resolve or pulls nothing? In that case again we have something that may be empty now but resolve later. So, for now using the much simpler "no sub-elements" rule, while recognizing that the edge case exists.
- Have no attributes that would be used as text content (such as `@alt` on the `<image>` element)

When a key definition has a child `<topicmeta>` element, content from that `<topicmeta>` element is used to determine effective content. Effective content from the key definition becomes the element content, with the following exceptions:

- For empty `<image>` elements, effective content is used as alternate text, equivalent to creating an `<alt>` sub-element to hold that content.
- For empty `<link>` elements, effective content is used as link text, equivalent to creating an `<linktext>` sub-element to hold that content.
- For empty `<link>` and `<xref>` elements, a key definition can be used to provide a short description in addition to the normal effective content. If the key definition includes `<shortdesc>` inside of `<topicmeta>`, that `<shortdesc>` should be used to provide effective content for a `<desc>` sub-element.

**Comment by robander on 29 July 2014**
Original DITA 1.2 text for this was: "For <link> tags with a @keyref attribute, the contents of the <shortdesc> tag in the key-defining element should provide the <desc> contents." This restatement expands that to <xref> which I believe is the intent. It also tries to stay consistent by using the term "effective content". Hopefully it is as clear / clearer.

- The <longdescref> and <longquoteref> elements are empty elements with no effective content. Key definitions are not used to set effective text for these elements.
- The <param> element does not have any effective content, so it does not pull any content.
- The <indextermref> element is not completely defined, so determining effective content for this element is also left undefined.
- The <abbreviated-form> is an empty element with special rules that determine effective content.

Effective text content is determined using the following set of rules:

1. For the <abbreviated-form> element, see the rules described in *<abbreviated-form>*
2. If an element does not allow the @href attribute, content is taken from the first <keyword> or <term> element inside of <keywords> inside of the <topicmeta>. For example, given the following key definition, empty <keyword>, <term>, and <dt> elements with the attribute keyref="nohref" would all result in the matching content "first":

```
<keydef keys="nohref">
  <topicmeta>
    <keywords><keyword>first</keyword><keyword>second</keyword><term>third</term></keywords>
  </topicmeta>
</keydef>
```

3. For elements that do allow @href, elements from within <topicmeta> that are legal within the element using @keyref are considered matching text. For example, the <xref> element allows @href, and also allows both <keyword> and <term> as children. Using the code sample from the previous item, an empty <xref> with keyref="test" would use all three of these elements as text content; after processing, the result would be equivalent to:

```
<xref keyref="test"><keyword>first</keyword><keyword>second</keyword><term>third</term></xref>
```

4. Otherwise, if <linktext> is specified inside of <topicmeta>, the contents of <linktext> are used as the effective content.

   **Note:** Because all elements that pull text will eventually look for content in the <linktext> element, using <linktext> for effective content is a best practice for cases where all elements getting text from a key definition should result in the same value.
5. Otherwise, if the element with the key reference results in a link, normal link text determination rules apply as they would for <xref> (for example, using the <navtitle> or falling back to the URI of the link target).

**Comment by robander on July 29 2014**
I think from close review that we should have a rule in here that is specific to elements allowed inside of <topicmeta> -- that is, author, source, data, and data-about. If those exist as a child of <topicmeta> in the key definition, then they are used to provide the content for the same elements with @keyref. I think (but am not sure) that this is the correct interpretation of the following line in the DITA 1.2 spec: "When a key definition has a <topicmeta> subelement, elements that refer to that key and that are empty may get their effective content from the first matching subelement of the <topicmeta> subelement of the key-defining topicref." I've left it out until others confirm.

When the effective content for a key reference element results in invalid tagging, that tagging should be generalized. For example, <linktext> in the key definition may use a domain specialization of <keyword> that is not valid in the key reference context, in which case the specialized element should be generalized to <keyword>. If the generalized content is also not valid, a text equivalent should be used instead. For example, <linktext>

may include <ph> or a specialized <ph> in the key definition, but neither of those are valid as the effective content for a <keyword>. In that case, the text content of the <ph> should be used.

**Comment by robander on 29 July 2014**
This is a change from DITA 1.2, but one that is needed to avoid nonsensical results. The 1.2 language was: "Elements within <linktext> that do not match the content model of the key reference directly or after generalization should be skipped. " In this case, the following key definition would give varying results:

```
<keydef keys="ph">
  <topicmeta>
    <linktext>This is <b>IMPORTANT</b></linktext>
  </topicmeta>
</keydef>
```

- In contexts where  is legal, the effective content would be "This is IMPORTANT".
- In contexts where <ph> is legal but there is no highlight domain, the effective content would be "This is <ph>IMPORTANT</ph>"
- Problem: in contexts where <ph> is not legal (such as when creating effective text for <keyword>), a strict interpretation of the spec would drop the  or <ph> element, and mean the effective content is "This is "

# Examples of keys

This section of the specification contains examples and scenarios. They illustrate a wide variety of ways that keys can be used.

## Examples: Key definition

Either <topicref> or <keydef> elements can be used to define keys.

In the following example, a <topicref> element is used to define a key; the <topicref> element also contributes to the navigation structure.

```
<map>
  <!--... -->
  <topicref keys="apple-definition" href="apple-gloss-en-US.dita" />
  <!--... -->
</map>
```

The apple-gloss-en-US.dita topic is processed as if the @keys attribute is not present.

In the following example, a <keydef> element is used to define a key.

```
<map>
  <!--... -->
  <keydef keys="apple-definition" href="apple-gloss-en-US.dita"/>
  <!--... -->
</map>
```

Because the <keydef> element sets the default value of the @processing-role attribute to "resource-only", the key definition does not contribute to the map navigation structure; it only serves as a key definition for "apple-definition".

## Examples: Key definitions for variable text

Key definitions can be used to store variable text, such as product names and user-interface labels. Depending on the key definition, the rendered output might have a link to a related resource.

**Comment by Kristen Eberlein**

Robert Anderson and I have talked about whether "variable text" is the correct phrase to use. Another option we considered was "generated text," but I think that has too many connotations about text that is generated by style sheets.

This example overlaps significantly with the original "Swap out variable content" example. What that example showed that is not covered here is:

- Using a `<linktext>` element to hold the variable text
- Authors using different key definitions to "swap out" content

In the following example, a "product-name" key is defined. The key definition contains a child `<keyword>` element nested within a `<keydef >`element.

```
<map>
  <keydef keys="product-name">
    <topicmeta>
     <keywords>
       <keyword>Thing-O-Matic</keyword>
     </keywords>
    </topicmeta>
  </keydef>
</map>
```

A topic can reference the "product-name" key using the following markup:

```
<topic id="topicid">
  <p><keyword keyref="product-name"/> is a product designed to ...</p>
</topic>
```

When processed, the output contains the text "Thing-O-Matic is a product designed to ... ".

In the following example, the key definition contains both a reference to a resource and variable text.

```
<map>
  <keydef keys="product-name" href="thing-o-matic.dita">
    <topicmeta>
     <keywords>
       <keyword>Thing-O-Matic</keyword>
     </keywords>
    </topicmeta>
  </keydef>
</map>
```

When processed using the key reference from the first example, the output contains the "Thing-O-Matic is a product designed to ... " text. The phrase "Thing-O-Matic" also is a link to the `thing-o-matic.dita` topic.

## Example: Scoped key definitions for variable text

Scoped key definition can be used for variable text. This enables you to use the same DITA topic multiple times in a DITA map, and in each instance the variable text can resolve differently.

The Acme Tractor Company produces two models of tractor: X and Y. Their product manual contains sets of instructions for each model. While most maintenance procedures are different for each model, the instructions for changing the oil are identical for both model X and model Y. The company policies call for including the specific model number in each topic, so a generic topic that can be used for both models is not permitted.

1. The authoring team references the model information in the `changing-the-oil.dita` topic by using the following mark-up:

```
<keyword keyref="model"/>
```

2. The information architect examines the root map for the manual, and decides how to define key scopes. Originally, the map looks like the following:

```
<map>
  <!-- Model X: Maintenance procedures -->
  <topicref href="model-x-procedures.dita">
    <topicref href="model-x/replacing-a-tire.dita"/>
    <topicref href="model-x/adding-fluid.dita"/>
  </topicref>

<!-- Model Y: Maintenance procedures -->
  <topicref href="model-y-procedures.dita">
    <topicref href="model-y/replacing-a-tire.dita"/>
    <topicref href="model-y/adding-fluid.dita"/>
  </topicref>
```

3. The information architect wraps each set of procedures in a `<topicgroup>` element and sets the `@keyscope` attribute.

```
<map>
  <!-- Model X: Maintenance procedures -->
  <topicgroup keyscope="model-x">
    <topicref href="model-x-procedures.dita">
      <topicref href="model-x/replacing-a-tire.dita"/>
      <topicref href="model-x/adding-fluid.dita"/>
    </topicref>
  </topicgroup>

<!-- Model Y: Maintenance procedures -->
  <topicgroup keyscope="model-y">
    <topicref href="model-y-procedures.dita">
      <topicref href="model-y/replacing-a-tire.dita"/>
      <topicref href="model-y/adding-fluid.dita"/>
    </topicref>
  </topicgroup>
```

This defines the key scope for set of procedures.

4. The information architect then adds key definitions to each set of procedures, as well as a reference to the `changing-the-oil.dita` topic.

```
<map>
  <!-- Model X: Maintenance procedures -->
  <topicgroup keyscope="model-x">
    <keydef keys="model">
      <topicmeta>
        <linktext>X</linktext>
      </topicmeta>
    </keydef>
    <topicref href="model-x-procedures.dita">
      <topicref href="model-x/replacing-a-tire.dita"/>
      <topicref href="model-x/adding-fluid.dita"/>
      <topicref href="common/changing-the-oil.dita"/>
    </topicref>
  </topicgroup>

<!-- Model Y: Maintenance procedures -->
  <topicgroup keyscope="model-y">
    <keydef keys="model">
      <topicmeta>
        <linktext>Y</linktext>
      </topicmeta>
    </keydef>
    <topicref href="model-y-procedures.dita">
      <topicref href="model-y/replacing-a-tire.dita"/>
      <topicref href="model-y/adding-fluid.dita"/>
```

```
      <topicref href="common/changing-the-oil.dita"/>
    </topicref>
  </topicgroup>
```

When the DITA map is processed, the `changing-the-oil.dita` topic is rendered twice. The model variable is rendered differently in each instance, using the text as specified in the scoped key definition.

## Example: Duplicate key definitions within a single map

A DITA map might contain duplicate key definitions. How processors find the effective key definition depends on document order and whether conditional processing occurs.

In the following example, a map contains two definitions for the key "load-toner":

```
<map>
  <!--... -->
  <keydef keys="load-toner" href="model-1235-load-toner-proc.dita"/>
  <keydef keys="load-toner" href="model-4545-load-toner-proc.dita"
  />
  <!--... -->
</map>
```

In this example, only the first key definition -- in document order -- of the "load-toner" key is effective. All references to the key within the scope of the map resolve to the topic `model-1235-load-toner-proc.dita`.

In the following example, a map contains two definitions for the "file-chooser-dialog" key; each key definition specifies a different value for the `@platform` attribute.

```
<map>
  <!--... -->
  <keydef keys="file-chooser-dialog" href="file-chooser-osx.dita" platform="osx"/>
  <keydef keys="file-chooser-dialog" href="file-chooser-win7.dita" platform="windows7"/>
  <!--... -->
</map>
```

In this case, the effective key definition is determined not only by the order in which the definitions occur, but also by whether the active value of the `@platform` attribute is "osx" or "windows7". Both key definitions are *potentially* effective because they have distinct values for the conditional attribute. Note that if no active value is specified for the `@platform` attribute at processing time, then both of the key definitions are present and so the first one in document order is the effective definition.

If the DITAVAL settings are defined so that both "osx" and "windows" values for the `@platform` attribute are excluded, then neither definition is effective and the key is undefined. That case can be avoided by specifying an unconditional key definition after any conditional key definitions, for example:

```
<map>
  <!--... -->
  <keydef keys="file-chooser-dialog" href="file-chooser-osx.dita" platform="osx"/>
  <keydef keys="file-chooser-dialog" href="file-chooser-win7.dita" platform="windows7"/>
  <keydef keys="file-chooser-dialog" href="file-chooser-generic.dita"/>
  <!--... -->
</map>
```

If the above map is processed with both "osx" and "windows" values for the `@platform` attribute excluded, then the effective key definition for "file-chooser-dialog" is the `file-chooser-generic.dita` resource.

## Example: Duplicate key definitions across multiple maps

A root map might reference submaps, each of which might contain duplicate key definitions. Which key definition is effective depends on the document order.

In the following example, a root map contains a key definition for the "toner-specs" and references to two submaps.

```
<map>
  <keydef keys="toner-specs" href="toner-type-a-specs.dita"/>
  <mapref href="submap-01.ditamap"/>
  <mapref href="submap-02.ditamap"/>
</map>
```

The first submap, `submap-01.ditamap`, contains key definitions for the "toner-specs" and "toner-handling":

```
<map>
  <keydef keys="toner-specs" href="toner-type-b-specs.dita"/>
  <keydef keys="toner-handling" href="toner-type-b-handling.dita"/>
</map>
```

The second submap, `submap-02.ditamap`, contains key definitions for the "toner-specs", "toner-handling", and "toner-disposal":

```
<map>
  <keydef keys="toner-specs" href="toner-type-c-specs.dita"/>
  <keydef keys="toner-handling" href="toner-type-c-handling.dita"/>
  <keydef keys="toner-disposal" href="toner-type-c-disposal.dita"/>
</map>
```

For this example, the effective key definitions are listed in the following table.

| Key | Bound resource |
|---|---|
| toner-specs | `toner-type-a-specs.dita` |
| toner-handling | `toner-type-b-handling.dita` |
| toner-disposal | `toner-type-c-disposal.dita` |

The key definition for "toner-specs" in the root map is effective, because it is the first encountered in a breadth-first traversal of the map tree. The key definition for "toner-handling" in `submap-01.ditamap` is effective, because submap-01 is included before submap-02 and so comes first in the map tree. The key definition for "toner-disposal" is effective because it is the only definition of the key in the map tree.

## Example: Scoped key references

You can address scoped keys definitions from outside the key scope where the key definitions are specified.

```
<map xml:lang="en">
  <title>Examples of scoped key references</title>

  <!-- Key scope #1 -->
  <topicgroup keyscope="scope-1">
    <keydef keys="key-1" href="topic-1.dita"/>
    <topicref keyref="key-1"/>
    <topicref keyref="scope-1.key-1"/>
    <topicref keyref="scope-2.key-1"/>
  </topicgroup>

  <!-- Key scope #2 -->
  <topicgroup keyscope="scope-2">
    <keydef keys="key-1" href="topic-2.dita"/>
    <topicref keyref="key-1"/>
    <topicref keyref="scope-1.key-1"/>
    <topicref keyref="scope-2.key-1" />
  </topicgroup>

  <topicref keyref="key-1" />
```

```
  <topicref keyref="scope-1.key-1" />
  <topicref keyref="scope-2.key-1" />

</map>
```

For this example, the effective key definitions are listed in the following tables.

*Table 1: Effective key definitions for scope #1*

| Key reference | Resource |
|---|---|
| key-1 | `topic-1.dita` |
| scope-1.key-1 | `topic-1.dita` |
| scope-2.key-1 | `topic-2.dita` |

*Table 2: Effective key definitions for scope #2*

| Key reference | Resource |
|---|---|
| key-1 | `topic-2.dita` |
| scope-1.key-1 | `topic-1.dita` |
| scope-2.key-1 | `topic-2.dita` |

*Table 3: Effective key definitions for root map*

| Key reference | Resource |
|---|---|
| key-1 | Undefined |
| scope-1.key-1 | `topic-1.dita` |
| scope-2.key-1 | `topic-2.dita` |

**Comment by Kristen Eberlein**

The comments in the original code block includes the following: "Keys defined in a scope become a part of the parent scope, prefixed with the scope name and a period."

Do they really become part of the parent scope? Or is it just that they can be addressed from within the parent scope?

We've use the terminology "parent scope" and "child scope". Do we need "peer scope"?

## Example: Key definitions in nested key scopes

A root map might contain nested key scopes, each of which might contain duplicate key definitions. Which key definition is effective depends on key-scope precedence rules.

Consider the following DITA map:

```
<map>
  <title>Root map</title>
  <!-- Root scope -->
  <keydef keys="a"/>

  <!-- Key scope A -->
  <topicgroup keyscope="A">
    <keydef keys="b"/>
```

```
     <!-- Key scope A-1 -->
    <topicgroup keyscope="A-1">
      <keydef keys="c"/>
    </topicgroup>

     <!-- Key scope A-2 -->
    <topicgroup keyscope="A-2">
      <keydef keys="d"/>
    </topicgroup>
  </topicgroup>

  <!-- Key scope B -->
  <topicgroup keyscope="B">
    <keydef keys="a"/>
    <keydef keys="e"/>

  <!-- Key scope B-1 -->
    <topicgroup keyscope="B-1">
      <keydef keys="f"/>
    </topicgroup>

  <!-- Key scope B-1 -->
    <topicgroup keyscope="B-2">
      <keydef keys="g"/>
    </topicgroup>
  </topicgroup>
</map>
```

The key scopes in this map form a tree structure.

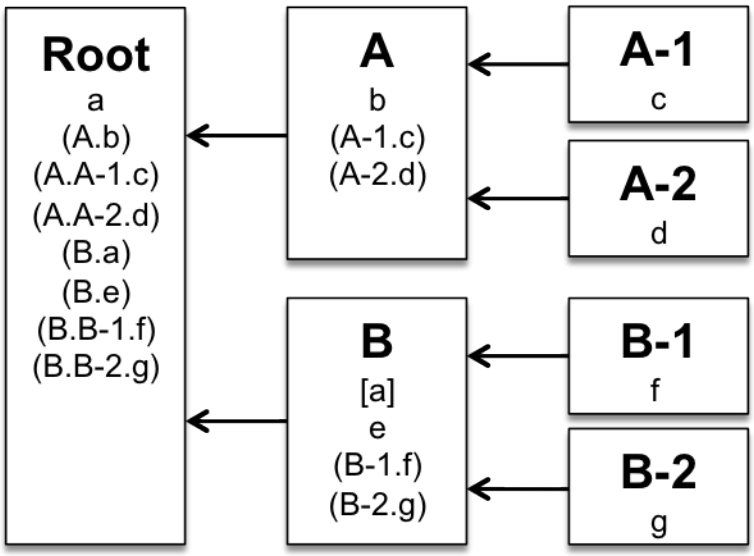


**Figure 1: Graphical representation of the key scopes**

Each box in the diagram represents a key scope; the name of the key scope is indicated in bold with upper-case letters. Below the name of the key scope, the key definitions that are present in the scope are listed. Different typographic conventions are used to indicate where the key definition occurs:

**No styling**

The key definition occurs in the immediate key scope and is not overriden by a key definition in a parent scope. For example, key "a" in the root map.

**Parentheses**

The key definition occurs in a child scope. For example, keys "A-1.c" and "A-2.d" in key scope A.

**Brackets**

The key definition occurs in the immediate key scope, but it is override by a key definition in a parent scope. For example, key "a" in key scope B.

Arrows points from child to parent scopes.

Assume that each key scope contains numerous key references. For this example, effective key definitions for certain key scopes are listed in the following tables.

*Table 4: Key scope A-2*

| Key reference | Key |
|---|---|
| a | "a", defined in the root map |
| d | "d", as defined in the immediate key scope |
| A-2.d | "d", as defined in the immediate key scope |
| c | Undefined |
| A-1.c | "A-1.c", as defined in key scope A-1. This key name is available because it exists in the parent scope, key scope A. |
| A.A-1.c | "A-1.c", as defined in key scope A-1. This key name is available because it exists in the root key scope. |

*Table 5: Key scope B*

| Key reference | Key |
|---|---|
| e | "e", defined in the immediate key scope |
| a | "a", as defined in the *root key scope*. (While a key definition for "a" exists in the immediate key scope, it is overriden by the key definition that occurs in the parent key scope.) |
| B.a | "a", as defined in the *immediate key scope*. However, since a key definition for "a" exists in the parent key scope, the effective value for "a" in the immediate key scope still is overriden by the key definition that occurs in the parent key scope. |
| g | Undefined |
| B-2.g | "g", as defined in key scope B-2. |

## Example: Link redirection

This scenario outlines how different authors can redirect links to a common topic by using key definitions. This could apply to either `<xref>` or `<link>` elements.

A company wants to use a common DITA topic for information about recycling: `recycling.dita`. However, the topic contains a cross-reference to a topic that needs to be unique for each product line; each such topic contains product-specific URLs.

1. The editing team creates a `recycling.dita` topic that includes a cross-reference to the product-specific topic. The cross reference is implemented using a key reference:

   ```
   <xref keyref="product-recycling-info" href="generic-recycling-info.dita"/>
   ```

   The value of the `@href` attribute provides a fallback in the event that a product team forgets to include a key definition for "product-recycling-info".
2. Each product documentation group creates a unique key definition for "product-recycling-info". Each group authors the key definition in a DITA map, for example:

   ```
   <map>
     <!-- ... -->
     <keydef keys="product-recycling-info" href="acme-server-recycling.dita"/>
     <!-- ... -->
   </map>
   ```

   Each team can use the `recycling.dita` topic, and the cross reference in the topic resolves differently for each team.
3. A year later, there is an acquisition. The newly-acquired team wants to reuse Acme's common material, but it needs to direct its users to an external Web site that lists the URLs, rather than a topic in the product documentation. Their key definitionlooks like the following:

   ```
   <topicref  keys="product-recycling-info"
                    href="http://www.acme.com/server/recycling"
                    scope="external"/>
   ```

   When newly-acquired team uses the `recycling.dita` topic, it resolves to the external Web site; however for all other teams, the cross reference in the topic continues to resolves to their product-specific topic.
4. A new product team is formed, and the team forgets to include a key definition for "product-recycling-info" in one of their root maps. Because the cross reference in the `recycling.dita` topic contains a value for the `@href` attribute, the link falls back to `generic-recycling-info.dita`, thus avoiding a broken cross reference in the output.

## Example: Link modification or removal

This scenario outlines how different authors can remove or modify a `<link>` element in a shared topic.

A company wants to use a shared topic for information about customer support. For most products, the shared topic should include a link to a topic about extended warranties. But a small number of products do not offer extended warranties.

1. Team one creates the shared topic: `customer-support.dita`. The topic contains the following mark-up:

   ```
    <related-links>
     <link keyref="extended-warranties" href="common/extended-warranties.dita"/>
    </related-links>
   ```
2. The teams that need the link to the topic about extended warranties can reference the `customer-support.dita` topic in their DITA maps. When processed, the related link in the topic resolves to the `common/extended-warranties.dita` topic.
3. The teams that do not want the related link to the topic about extended warranties can include a key definition that in their DITA map that does not include an `@href` attribute, for example:

   ```
   <map>
     <!-- ... -->
     <keydef keys="extended-warranties"/>
     <!-- ... -->
   </map>
   ```

When processed, the related link in the topic is not rendered.

4. Yet another team wants to simply have a paragraph about extended warranties printed. They define the key definition for "extended-warranties" as follows:

```
<map>
  <!-- ... -->
  <keydef keys="extended-warranties"/>
    <topicmeta>
      <linktext>This product does not offer extended warranties.</linktext>
    </topicmeta>
  <!-- ... -->
</map>
```

When this team renders their content, there is no hyperlink in the output, just the "The product does not offer extended warranties" statement.

**Comment by Kristen Eberlein**

This does not work in DITA-OT 1.8.4.

## Example: Links from <term> or <keyword> elements

The `@keyref` attribute enables authors to specify that references to keywords or terms in a DITA topic should be rendered as a link to an associated resource.

In this scenario, a company with well-developed glossary wants to ensure that instances of a term that is defined in the glossary always include a link to the glossary topic.

1. An information architect adds values for the `@keys` attribute to all the of the `<topicref >`elements that are in the DITA map for the glossary, for example:

```
<map>
  <title>Company-wide glossary</title>
  <topicref keys="term-1" href="term-1.dita"/>
  <topicref keys="term-2" href="term-2.dita"/>
  <topicref keys="term-3" href="term-3.dita"/>
  <topicref keys="term-4" href="term-4.dita"/>
</map>
```

2. When authors refer to a term in a topic, they use the following mark-up:

```
<term keys="term-1"/>
```

When rendered, the content is pulled from the `<title>` element of the glossary topic. It also is a link to the glossary topic.

## Example: conref redirection

The `@conkeyref` attribute enables authors to share DITA topics that reuse content; it enables each author to specify a different key definition for common keys.

In this scenario, Acme produces content for product that is also resold through a business partner. When the DITA content is published for the partner, several items must be different, including the following:

- Product names
- Standard notes that contain admonitions

Simply using the `@conref` attribute would not be possible for teams that use a component content management system where every DITA topic is addressed by a globally-unique identifier (GUID).

1. Authors reference the reusable content in their topics by using the `@conkeyref` attribute, for example:

```
<task>
  <title><keyword conkeyref="reuse/product-name"/> prerequisites</title>
  <!-- ... -->
  <note conkeyref="warning-1/>
  <!-- ... -->
</task>
```

2. Authors create two different topics; one topic contain elements that contain the text that is appropriate for Acme, and the other topic contains elements that contain the text that is appropriate for the partner. Note that each reuse topic must contain the same DITA tags and values for the `@id` attribute, for example:

```
<topic id="reuse-oem">
  <title>Reuse topic for OEM partner</title>
  <body>
    <!-- ... -->
    <note id="warning-1"/>
    <p><keyword id="product-name"/></p>
    <!-- ... -->
  </body>
</topic>
```

3. The two versions of the DITA maps each contain different key definitions for "reuse". (This associates a key with the topic that contains the appropriate reusable elements.) For example:

```
<map>
  <!-- ... -->
  <keydef keys="reuse" href="acme-reuse.dita"/>
  <!-- ... -->
</map>
```

**Figure 2: DITA map for Acme**

```
<map>
  <!-- ... -->
  <keydef keys="reuse" href="oem-reuse.dita"/>
  <!-- ... -->
</map>
```

**Figure 3: DITA map for OEM partner**

When the DITA maps are published, the elements that are referenced by `@conkeyref` will use the reuse topic that is referenced by the `<keydef >` element in the map. The product names and warnings will be different in the output.

## Example: Key scopes and omnibus publications

Key scopes enable you to create omnibus publications that include multiple submaps that have common keys.

In this scenario, a training organization wants to produce a deliverable that includes all of their training course materials. Each course manual uses common keys for standard parts of the course materials, including "prerequisites," "overview", "assessment", and "summary.

An information architect creates a DITA map that contains the following mark-up:

```
<map xml:lang="en">
  <title>Training courses</title>
  <mapref href="course-1.ditamap"/>
  <mapref href="course-2.ditamap"/>
  <mapref href="course-3.ditamap"/>
  <topicref href="omnibus-summary.dita"/>
</map>
```

Each of the submaps contain `<topicref>` elements that reference resources using the `@keyref` attribute. Each submap uses common keys for standard parts of the course materials, including "prerequisites," "overview", "assessment", and "summary", and their key definitions resolve the key names to course-specific resources. For example:

```
<map xml:lang="en">
  <title>Training course #1</title>
  <mapref href="course-1/key-definitions.ditamap/>
  <topicref keyref="prerequisites"/>
  <topicref keyref="overview"/>
  <topicref keyref="assessment"/>
  <topicref keyref="summary"/>
</map>
```

Without using key scopes, the effective key definitions for the common keys resolve to those found in `course-1.ditamap`. This is not the desired outcome. By adding key scopes to the submaps, however, the information architect can ensure that the key references in the submaps resolve to the course-specific key definitions.

```
<map xml:lang="en">
  <title>Training courses</title>
  <mapref href="course-1.ditamap" keyscope="course-1"/>
  <mapref href="course-2.ditamap" keyscope="course-2"/>
  <mapref href="course-3.ditamap" keyscope="course-3"/>
  <topicref href="omnibus-summary.dita"/>
</map>
```

The information architect does **not** set `keys="summary"` on the `<topicref>` element in the omnibus map. Doing so would mean that all key references to "summary" in the submaps would resolve to `omnibus-summary.dita`, rather than the course-specific summary topics. This is because key definitions located in parent scopes override those located in child scopes.

## Example: Keys and collaboration

Keys enable authors to collaborate and work with evolving content with a minimum of time spent reworking topic references.

**Comment by Kristen Eberlein**

In the 1.2 spec, this example was titled "Splitting or recombining targets." I think we need to find a more appropriate title, and I don't know if what I've used is the correct choice. The example really is about link redirection with a minimum of fuss ...

In this scenario, authors collaborate on a publication that includes content for a product that is in the early stages of development. The company documentation is highly-structured and uses the same organization for all publications: "Introduction," "Example," and "Reference."

1. Author one creates a submap for the new product information. She knows the structure that the final content will have, but she does not want to create empty topics for information that is not yet available. She decides to initially author what content is available in a single topic. When more content is available, she'll create additional topics. Her DITA map looks like the following:

   ```
   <map>
     <title>New product content</title>
     <topicref keys="1-overview 1-intro 1-example 1-reference" href="1-overview.dita"/>
   </map>
   ```

2. Author two knows that he needs to add a `<topicref>` to the "Example" topic that will eventually be authored by author one. He references the not-yet-authored topic by key reference:

```
<topicref keyref="1-example"/>
```

   His topic reference initially resolves to the `1-overview.dita` topic.
3. Author one finally gets the information that she was waiting on. She creates additional topics and modifies her DITA map as follows:

```
<map>
  <title>New product content</title>
  <topicref keys="1-overview href="1-overview.dita">
    <topicref keys="1-intro" href="1-intro.dita"/>
    <topicref keys="1-example" href="1-example.dita"/>
    <topicref keys="1-reference" href="1-reference.dita"/>
  </topicref>
</map>
```

   Without needing to make any changes to the content, author two's topic reference now resolves to the `1-example.dita` topic.