DITA 1.3 proposed feature 13041 Facility for key-based cross-deliverable addressing

Contents

DITA 1.3 proposed feature 13041 Facility for key-based cross-deliverable addressing

Defines the meaning and addressing implications for peer-scope map references that specify a key scope for the map reference: keys defined in peer maps that are bound to a key scope in the referencing map may be addressed as scope-qualified keys.

Date and version information

Include the following information:

- Phase 2 proposal submitted 28 April 2013
- Champion: Eliot Kimber
- Email discussion: *https://lists.oasis-open.org/archives/dita/201304/msg00050.html*

Original requirement or use case

From the DITA mail list, *Aug 29, 2011 7:11:59 am, Eliot Kimber*:

> Per the discussion on the phone, my requirement is to be able to point to a different root map in order to address keys in that key space as a separate key space. This is a different requirement from that addressed by Proposal 13004, which is about scoping keys within a single map tree.
>
> My original thinking was that being able to bind a key reference to any map would also address the within-map-tree scoping requirements addressed by 13004, that is, if I can address a map as a distinct key space, it doesn't matter whether it's inside or outside the currently-processed map tree.
>
> However, the minimum requirement is for cross-publication addressing, which means addressing a root map (representing a separate key space) and then a key within that key space so that the data as authored unambiguously represents a key-based relation between something in one publication (root map) and something in another publication (different root map).

Use cases

There is a general requirement to be able to author navigation links between two deliverables that are otherwise independent, meaning that they are represented by distinct root maps, processed separately, and so on. Typical use cases include:

- Citations of other publications where the citation should be a navigable link to a specific delivered instance of the publication (e.g., the PDF for the publication, the EPUB, etc.).
- Links to topics in the context of other deliverables. For example, a reference from a task topic in one publication to a reference topic packaged in a separate deliverable. This case occurs in the DITA specification itself when the *DITA Architecture Specification* and *DITA Language Reference* are published as separate deliverables. In that case, there are links from mentions of element types and attributes in the architecture specification to their reference entries in the language reference.
- Links to non-topic elements within topics in the context of other deliverables. For example, a reference to a figure or table published in a separate publication.

The general authoring requirement is to be able to author key references and key definitions in the context of one root map that resolve to resources as used in the context of other root maps.

There are two main variations on this use case:

1. The author needs to author a key reference that does not know or care what deliverable the target resource may be in for a given rendition of the topic or map that contains the reference. This case is typified by the DITA specification itself, where the references to language reference entries from the architecture specification topics will be within-deliverable or cross-deliverable references depending on which packaging of the DITA specification is being produced (all-in-one or separate architecture specification and language reference specification.
2. The author needs to author a key reference with the intent (at the time of authoring) that the reference be to a resource that is known to be (and expected to be) in a separate publication. This case is typified by a set of publications that are, for whatever reason, always published as separate deliverables, for example, different standards, manuals for separate products, books by different authors, etc.

In case (1) the initial key reference cannot itself specify anything about the specific root map or deliverable that the target resource will be in because the containing root map for the target resource is unknowable by the author of the key reference (which is the general case) or is known to vary, as in the case of the DITA specification itself. This means that any solution cannot, for example, depend on a separate specification of the target root map as part of the initial key reference.

In case (2) the initial key reference could specify the specific root map, as it is the author's intent to address a specific map. But this could still be problematic in terms of doing *deliverable*-specific addressing and would of course preclude using the link in a case-1 scenario should the usage of the content change in the future.

Thus for both cases there is an implicit requirement that any binding of keys to root maps or deliverables must be done as part of the key *definition*, not the initial key reference, so that the referenced key may be bound to different key spaces or scopes as required for a given delivery of the map or topic that contains the reference.

Proposed solution

There are two aspects to this proposal:

- A specific meaning applied to peer-scope map references that also specify key scopes. This aspect builds on and extends proposal 13004, scoped keys.
- An abstract processing model that uses generated key definitions to establish the targets of cross-deliverable links for specific delivered versions of the root maps involved.

Terminology

In this proposal the following terms are used with the following meanings:

root map	A DITA map that is the input to a process that produces a *deliverable*.
deliverable	The result of processing a root map to produce an output reflecting a unique set of input parameters including the deliverable data type (HTML, PDF, EPUB, etc.), the filtering specs (DITAVAL files), the delivered location (e.g., URL of where the deliverable will be published), and any other process-specific parameters what would result in a different deliverable (in particular, parameters that determine processor behavior where the DITA spec allows different behaviors). May also use the term *deliverable instance* to distinguish from *deliverable definition*.
deliverable definition	The set of parameters used to produce a deliverable. Deliverables are identified by the combination of root map and deliverable definition, meaning that a given root map processed using a given deliverable definition will always produce the same deliverable instance.
location as authored	The location of a DITA-addressible resource in its source format, e.g., a topic element, a map, a non-topic element

	within a topic, an image bound to a key, etc. Locations as authored are governed by the DITA standard and addressed using DITA-defined addressing facilities (key references and URIs).
location as delivered	The location of a DITA-addressible resource as it is reflected in a deliverable instance. The details of deliverable locations are determined by the format and semantics of the deliverable and, possibly, by the delivery access mechanism (e.g., reading tool, browser, etc.) used to access the deliverable. For example, an HTML deliverable would use URLs and HTML ID references, while a PDF would use PDF page locations or named anchors.
production manager	A DITA processor that manages the generation of deliverables and that has full knowledge of all peer resources that may be involved in cross-deliverable linking and full knowledge of all relevant deliverables that have been produced. For example, a component content management system could act as a production manager.

All other DITA-specific terms are intended to have the meanings as defined in either the DITA 1.2 specification or in proposal 13004.

The meaning of peer-scope map references (extension to 13004)

In order to enable cross-deliverable addressing, proposal 13004 is extended as follows:

Maps that are referenced with a @scope value of "peer" do not contribute to the key space of the referencing map, meaning that peer maps are treated as distinct and completely separate key spaces.

When a map is referenced with a @scope of "peer" and the topic reference also specifies a key scope (@keyscope), key references to that scope are taken to be references to the keys as defined in the peer map. Such references represent peer cross-deliverable references.

Note: While the author's intent in this case is clear because all the resources involved are unambiguously addressed through the key references and key definitions involved, the delivered result of such references is entirely processor dependent.

Conforming DITA processors are not required to support the resolution of or delivery of cross-deliverable references. Note, however, that the general implication of peer scope is that the peer resources are all available in the same processing or management context. This means that processors always have the potential to resolve peer key references when the peer resources are in fact managed together. There may, of course, be performance, scale, and user interface challenges in implementing such systems, but the fundamental ability to resolve any given reference is ensured when the source files are physically accessible. That is, given a correctly-specified key definition to a peer map, if the map itself is accessible it must then be possible to process that map in order to resolve keys defined by it, if it is also possible to resolve keys in the map making the peer map reference.

Note also the inverse implication: if the peer resource is not available then it is necessarily impossible to resolve the key reference. For that reason, processors must be prepared to provide appropriate messages or other indicators when a peer resource cannot be resolved for a particular processing purpose. Depending on how a given set of DITA resources is authored, managed, and processed, peer resources may be resolvable or not at different points in the content's life cycle.

For this example, there are two root maps: Map A and Map B.

Map A references Map B as a peer map and binds the scope name "map-b" to it:

```
<map><title>Map A</title>
```

```
<mapref
  href="../map-b/map-b.ditamap"
  scope="peer"
  keyscope="map-b"
/>
  ...
</map>
```

Map B defines keys normally:

```
<map><title>Map B</title>
<keydef
  keys="topic-b-1"
  href="topics/b-1.dita"
/>
<keydef
  keys="topic-b-1"
  href="topics/b-1.dita"
/>
 ...
</map>
```

A topic used by Map A may use the scope name "map-b" to explicitly reference a key defined in Map B:

```
<topic id="a-1"><title>A 1</title>
<body>
<p>See <xref keyref="map-b.topic-b-1">B 1</xref> ...</p>
</body>
</topic>
```

For the reference from topic "a-1", the key reference is taken to be a reference to the key name "topic-b-1" in the scope "map-b". Because the peer map Map B has been bound to the scope name "map-b" in the context of root map Map A, the key is resolved to the key "topic-b-1" as defined in root map Map B, which resolves to the topic in document "map-b/topics/b-1.dita".

Figure 1: Example—Peer map reference and scoped key reference

The referenced peer map may specify @keyscope on its root element. In that case, the @keyscope on the peer map is ignored for the purpose of resolving scoped key references from the referencing map. This avoids the need for processors to have access to or examine the peer map before being able to resolve scope key references.

Note: For example, given this map reference in map Map A:

```
<mapref
  keyscope="scope-a"
  scope="peer"
  href="map-b.ditamap"
/>
```

Where "map-b.ditamap" is:

```
<map keyscope="scope-b">
 ...
</map>
```

For references from the context of Map A, key references of the form "scope-a.somekey" will be resolved to keys defined in the global scope of map B. The presence or absence of a @keyscope attribute on Map B's <map> element has no effect in this case. Another way to view this is that a key reference to the scope "scope-b.somekey" is equivalent to the unscoped reference "somekey" processed in the context of Map B as the root map. In both cases, the presence of @keyscope on Map B's <map> element has no effect, in the first case (peer key reference) because it is explicitly ignored, in the second because the key reference will necessarily be within the scope "scope-b" and therefore does not need to be scope qualified.

For local-scope map references, the @keyscope attribute values on the map reference and the map element are merged, meaning that both the scopes named on the map reference and on the map will be resolvable for key references processed in the context of the referencing root map. The referenced map does not create another level of scope hierarchy when referenced as a sub map.

Note: This rule ensures that the same key reference (e.g., "scope-b.somekey") will resolve to the same resource whether the referenced map is referenced as a local map (sub map) or a peer map.

Abstract processing model for cross-deliverable links

Given a map, Map A, that has a link to a resource in a peer map, Map B, and given knowledge of any deliverable instance produced from Map B, it is possible to construct a normal key definition with a @scope value of "external" that addresses the location of the resource as delivered.

For example, if the link is to topic "b-1" in Map B using the key "map-b.topic-b-1", and the topic is rendered to HTML as "`http://example.com/maps/map-b/topics/b-1.html`", then the key definition for use by Map A would be:

```
<keydef keys="map-b.topic-b-1"
  scope="external"
  format="html"
  href="http://example.com/maps/map-b/topics/b-1.html"
/>
```

Any existing key-aware DITA 1.2 processor should be able to process this key definition and produce a working link.

Thus, it is possible to use sets of keys with external scope to define the mapping from keys as defined in a given root map to the deliverable-specific locations of the resources bound to those keys.

However, it would not, in the general case, be possible to author cross-deliverable links this way for several reasons:

1. In the general case, it is impossible to know the location of any resource as delivered until the deliverable has been generated and there is no guarantee that any particular deliverable will have been created at the time of authoring.
2. There may be multiple delivered forms of the target resource and therefore there needs to be at least one key definition per deliverable form of each linked resource.
3. The set of possible deliverable forms of any given resource is both unbounded and unknowable at authoring time. Therefore it is impossible in the general case to author separate key bindings for each deliverable form of the target resource.

Thus, the only practical approach in the general case is to author the link to the resource as authored and let processors produce the deliverable-specific key bindings as needed and as the necessary knowledge is available.

This proposal, combined with proposal 13120, Vocabulary for publishing process details, provides an abstract processing model and interchange representation for the information required by that processing model. This interchange representation can be used, for example, to allow different processors to work together to produce deliverables for different root maps, such as where different organizations or departments are producing documents that will link to each other when delivered but that are otherwise managed and processed independently. This abstract process definition does not require processors to implement this map-based approach literally. Processors that have full knowledge of all maps involved and all deliverables produced may choose to manage the necessary information and do the processing in any way that produces working and correct cross-deliverable links (meaning the links resolve to the appropriate target in the appropriate deliverable). Two processors given the same, or equivalent, deliverable definitions and root maps should produce the equivalent deliverables with respect to the cross-deliverable links, regardless of how they implement the processing.

In particular, the meaning of DITA addresses to content as authored is unambiguous and invariant and not subject to processor discretion.

Another important implication of using normal DITA maps for interchanging as-delivered key bindings is that existing DITA 1.2, key-aware processors should work normally given the appropriate deliverable-specific maps.

Thus, support for cross-deliverable addressing does not necessarily require implementation changes to existing DITA 1.2 processors simply in order to produce deliverables with cross-deliverable links. Any new processing involves the generation of the deliverable-specific maps, not their subsequent processing. However, because the deliverable-specific maps could be created manually, it is possible to implement this specification using only DITA 1.2 key-aware processing.

The general process for producing a deliverable from a root map that includes cross-deliverable links is as follows. In this process, the *initial root map* is the map for which a deliverable with working cross-deliverable links is desired.

1. Process the initial root map to determine what other root maps it has links to. These are the *peer map dependencies*.
2. For each peer map dependency, determine what deliverables are required. These are the *deliverable dependencies*. Deliverable dependencies are represented by root map/deliverable definition pairs.

   Note: The determination of the deliverable dependencies may be determined using facilities defined in proposal 13120, by business rules that define what deliverables are allowed or required to be linked to, or simply by the set of possible deliverables the processor is able to produce. For example, a processing system might maintain a set of deliverable definitions for each root map and identify one deliverable dependency for each deliverable definition.
3. For each deliverable dependency generate a new *deliverable dependency map*, containing only key definitions or <topicgroup> elements, where the key definitions reflect the location, as delivered, of each resource to which a key is bound in the input root map. The key definitions must have a @scope of "external" and a @format value appropriate for the deliverable (e.g., "html", "pdf", etc.). Where a key definition in the peer map as authored is within a relationship table, the key-defining topicref must be retained in the deliverable specific-map. The relationship table itself must not be retained in the deliverable-specific map. If the relationship table establishes any scopes those scopes must be represented in the deliverable-specific map using <topicgroup> elements. Each generated key definition should include a navigation title, per the general requirements for external-scope topic references. If the original key definition did not include a navigation title, the method for generating the navigation title is processor defined.

   Note: For example, relationship tables could be translated for use in deliverable-specific maps by simply changing all the relationship table elements to <topicgroup> elements, retaining any @keyscope attributes.

   Note: In the case where a map defines a normal-role topicref with a key that is then itself a reference to a resource-only key definition, the deliverable-specific map need not retain the indirection from the normal-role topicref to the resource-only key definition as long as the end result is that the normal-role key definition is bound to the correct location as delivered.

   Keys that are not bound to resources (where the key definition does not specify any @keyref or @href) should be unchanged in the deliverable-specific map.

   Note: How the processor determines the delivered location for a given key-to-resource binding is processor specific. It may require actually generating the deliverable or the processor may be able to determine the deliverable location without doing the actual deliverable generation.
4. Generate a copy of the initial root map where all the peer-scope map references are replaced with local-scope map references to the deliverable dependency maps. This is the *root deliverable map* and will be used to generate the deliverable for the initial root map. Note that this map is not the same as the deliverable dependency map for this same root map. The root deliverable map reflects the original map with only references to peer maps modified, while deliverable dependency maps reflect only key definitions from the dependency root map.

   Note: A single map reference from the initial root map may result in multiple map references in the deliverable-specific map.
5. As necessary, modify the root deliverable map or the deliverable-dependency maps to reflect the desired link-to-deliverable bindings.

   Note: For example, in the case where there are two deliverable-specific maps for a given peer dependency, one will necessarily be referenced first, which means that all of its key definitions will take precedence over the key definitions for the other deliverable. The key definitions could be adjusted by any or all of these means:

    - Change the order the deliverable-dependency maps are included from the deliverable-specific map.
    - Remove from the deliverable-dependency maps topicrefs that are not wanted.
    - Move or copy specific topicrefs from deliverable-dependency maps to the end of the root deliverable-specific map, where they will override any binds for the same key name in included maps. (Putting the key definitions at the end of the root map ensures that they do not override any overrides of the same key name that were already in the initial root map.). This last approach avoids the need to modify deliverable-dependency maps.

    The actual editing of these maps could, of course, be done manually by humans or could be done literally or logically by a processor based on business rules, user-supplied configuration, etc. This process is defined such that it is always possible for humans do implement it using only normal DITA 1.2 processors and editing tools.
6. Process the root deliverable map to produce the deliverable.

Practical implications of this process include:

- In the case where two maps used as initial root maps have links to each other, it may be necessary to perform steps 3 through 6 twice to ensure that the final form of any link did not cause the deliverable-specific locations of anything to change. For example, if the target deliverable is PDF and links are to specific pages, rather than to named anchors, the resolved link text for a cross-publication link could cause the pagination to change in the referencing PDF from the first pass to the second.

  In the worst case, delivery of any given initial root map requires producing the deliverables for all of the deliverable dependencies. However, in practice it should be possible to avoid this through various means, including choices for how deliverable-specific addresses are implemented, how link text is managed, and so on.

  You can also express this as: a change to any of the resources used in the context of any given initial root map may require regeneration of the deliverables for any other initial root map that has it as a dependency. In practice this means that either the addresses of deliverable versions need to be as insensitive as possible to details of the content or the production of sets of linked deliverables has to be carefully managed to ensure consistency across the deliverable set. The sensitivity of delivered locations is a function of both the details of the deliverable format and how things like filenames, IDs, and named anchors are generated. The DITA @copyto attribute can be used to define persistent filenames for use in deliverables where a single deliverable consists of multiple files, such as HTML.
- If deliverable-specific maps are not modified directly as described in Step (5), it should only be necessary to regenerate those maps when the either the source from which the deliverable was produced changes. This suggests that even in the case where there is production manager, authors can still maintain more or less persistent sets of deliverable-specific maps in order to simply and optimize deliverable production.
- Authoring tools may wish to provide the ability to see previews of resolved cross-deliverable links. Where authoring tools have access to the peer dependency maps they can choose to simply enable resolution of the links to the source as authored. They may further provide facilities for using deliverable-specific maps for resolving references to specific deliverables. This would likely require specific features for configuring knowledge of deliverable definitions so that, for example, relative references from one deliverable to another are resolvable in the context of the authoring environment. Authoring tools may also provide features for managing deliverable definitions, either as defined in proposal 13120 or using some proprietary deliverable definition mechanism. Where authoring tools do provide features for using deliverable definitions they should support at least the import of 13120-format delivery definitions. Where they provide features for creating and managing deliverable definitions they should support the export of 13120-format delivery definitions.
- Because deliverable definitions include filtering specifications, authoring and component content management systems that provide features for viewing DITA content with different filtering conditions applied, such systems may need to implement more sophisticated key space management features in order to optimize the resolution of cross-deliverable links as influenced by filtering conditions. That is, for a given cross-deliverable link, the ultimate destination as delivered and the final link text may be a function of both the target deliverable type and the specific filtering conditions in effect.
- Generating deliverable-specific maps requires some way to capture the deliverable-specific location of each key-bound resource in the input root map.

Because the output processor is generating the output it must have access to the final locations that each key-bound resource resulted in. However, many processors will not have been implemented to capture this mapping.

For processors that do not do capture the mapping, a non-trivial change or addition to the existing processing will likely be required in order to capture the mapping. While, in theory, a human could build such mappings by inspection after the fact, in practice this facility will only be generally useful if the deliverable-specific maps are generated automatically.

One potential simplifying technique is to implement a deterministic mechanism for generating result locations from source key definitions such that the final location can be determined simply by processing the source map without actually doing the final deliverable generation. For example, instead of using the XSLT generate-id() function to generate result IDs or anchors during final output processing, build the map of keys to output locations as a pre-process on the map and then provide the key-to-location mapping as input to the final generation process.

Benefits

Address the following questions:

- This feature benefits authors who need to create cross-deliverable links, including the DITA Technical Committee itself.
- This feature enables a form of linking that is not otherwise possible in DITA 1.2 using DITA-defined mechanisms.
- It is difficult to know how many people will use this feature but it is not an uncommon requirement. It is especially critical for publishing contexts where it is not usually or ever an option to combine otherwise separate publications into single deliverables simply to enable cross-publication linking, such as standards, traditional publishing, regulated environments, and so on.
- This feature has a significant positive impact for users who need it, making the otherwise impossible possible.

Technical requirements

DTD and Schema modifications	No modifications to existing vocabulary is required.
Processing impact	See the proposal details above. The processing impact falls primarily on the components that produce deliverables, in that there is now a requirement to be able to generate the mapping from source key definitions to their deliverable-specific locations. Given that mapping (and thus, deliverable-specific maps reflecting the mappings), then the resulting deliverable-specific maps can be processed normally by existing key-aware DITA 1.2 processors. Deliverable producers may need to add or refine features specific to the generation of cross-deliverable links for a specific deliverable type. For example, PDF generating processors may need to adjust how they handle PDF-to-PDF links. Likewise, delivery or reading tools that operate on DITA source may need to expand or add features for cross-deliverable linking. There are potential additional impacts on authoring and component content management tools that want to provide features for authoring, viewing, and managing cross-deliverable links. However, such features are not required by this proposal.

Overall usability	For map authors concerned only with authoring links, this facility adds little additional complexity beyond scoped keys in general. The complexity of managing deliverables with cross-deliverable links is inherent in the requirement for cross-deliverable links. This facility does not add complexity that was not already there. Any DITA user trying to do cross-deliverable links using DITA 1.2 was already faced with implementing some form of custom, one-off solution. Any user that does not need cross-deliverable links will not be affected by this feature.

Costs

Outline the impact (time and effort) of the feature on the following groups:

- Maintainers of the DTDs and XSD: None (no change to existing markup)
- Editors of the DITA specification:
  - How many new topics will be required?

    May require a new topic to specifically discuss cross-deliverable links (essentially the text of this proposal).
  - How many existing topics will need to be edited?

    Will require updates to the existing linking and addressing topics to ensure the meaning of "peer" for @scope on <mapref> is clearly and consistently described.

    Will require updates to the discussion of map references generally and the <mapref> element in particular to ensure that the meaning of "peer" for @scope is clearly and consistently described.
  - Will the feature require substantial changes to the information architecture of the DITA specification? If so, what?

    No changes to the existing information architecture are required beyond those required by scoped keys generally.
- Vendors of tools:

  See the discussion of practical implications given above. Processors that generate deliverables will be affected. Authoring and component content management tools will be affected to the degree they choose to support authoring, management, and viewing of cross-deliverable links. No new features are required for authoring and management tools.
- DITA community-at-large. Will this feature add to the perception that DITA is becoming too complex? Will it be simple for end users to understand?

  For users that have the requirement for cross-deliverable links the feature will be understandable. For users that do not have the requirement, it could be a challenge to understand as the subject itself is inherently challenging. Any new addressing feature always adds to the perception of increased complexity.

Examples

This example uses two maps, Map A and Map B, and uses them together in three configurations that serve to demonstrate the details of the addressing and processing. For the purposes of this example, the maps are stored relative to each other, each map within its own directory under a common root directory. For this example, two deliverables are generated: HTML and PDF. To keep the example simple there is no filtering or flagging involved.

Map A (map-a/map-a.ditamap):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <title>Map A</title>
```

```
    <!-- This map establishes the scope "map-a".

         The use case is that you want to be able to reference
         topics using keys of the form "map-a.foo" from outside
         this scope, but "foo" from within the scope, that is,
         topics that are only ever used from within this scope
         or for which "foo" will be defined in whatever scope
         the topic is used.

         We now include the map-a local keydefs within this
         scope so that these keydefs are within the scope
         and thus unqualified references to them will work.

         Note that we can't put the scope on the map with
         the keydefs because that map doesn't then contain
         the navigation topicrefs that we want to be in the
         scope.

         This means that each use of topic-a's.
      -->
  <topicgroup>
    <topicmeta>
      <navtitle>Keydefs Map A</navtitle>
    </topicmeta>
    <keydef
      keys="topic-a-root"
      href="topics/topic-a-root.dita"/>
    <keydef
      keys="topic-a-01"
      href="topics/topic-a-01.dita"/>
    <keydef
      keys="topic-a-02"
      href="topics/topic-a-02.dita"/>
  </topicgroup>
  <!-- Include Map B's map as a peer and assign a keyscope to it. -->
  <mapref
    href="../map-b/map-b.ditamap"
    scope="peer"
    keyscope="map-b"
  />
  <!-- Navigation tree for map A: -->
  <topicref
    keyref="topic-a-root">
    <topicref
      keyref="topic-a-01"/>
    <topicref
      keyref="topic-a-02"/>
  </topicref>
</map>
```

Map B (map-b/map-b.ditamap):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <title>Map B</title>
  <topicgroup>
    <topicmeta>
      <navtitle>Keydefs Map B scope="local"</navtitle>
    </topicmeta>
    <keydef
      keys="topic-b-root"
      href="../topics/topic-b-root.dita"/>
    <keydef
      keys="topic-b-01"
      href="../topics/topic-b-01.dita"/>
    <keydef
      keys="topic-b-02"
      href="../topics/topic-b-02.dita"/>
  </topicgroup>
  <mapref
    href="../map-a/map-a.ditamap"
    scope="peer"
    keyscope="map-a"
  />
  <topicref
    keyref="topic-b-root">
    <topicref
      keyref="topic-b-01"/>
    <topicref
      keyref="topic-b-02"/>
  </topicref>
</map>
```

Note that each map references the other map with @scope of "peer" and with a key scope. These peer map references bind their key scope names to all the keys defined in the referenced map. However, the keys defined in the referenced map are not added to the key space of the referencing map because of the "peer" value for @scope.

Map A includes topic topic-a-01 that has a scoped key reference to a key in the scope "map-b":

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">
<topic id="topic-a-01">
  <title>Topic A-01</title>
  <body>
    <p>This topic is "owned" by map A. The references to keys in the map A
 are
      unqualified. The references to keys in map B are qualified with the
 key
      scope "map-b". The "map-b" keyscope may be local to the root map that
 also
      includes map A or may be a peer map referenced with the assoicated
      keyscope "map-b". The key references are the same in both cases.</p>
    <p>Cross reference to topic A-02: <xref
      keyref="topic-a-02">A-02</xref></p>
    <p>Cross reference to topic B-01: <xref keyref="map-b.topic-b-01">B-01</
xref></p>
    <p>Cross reference to figure in topic A-02: <xref
      keyref="topic-a-02/fig-01"/></p>
    <p>Cross reference to figure in topic B-02: <xref
      keyref="map-b.topic-b-02/fig-01"/></p>
  </body>
</topic>
```

When this topic is processed in the context of Map A used as an initial root map the key reference "map-b.topic-b-01" resolves to the key "topic-b-01" defined key scope "map-b", which in this case is bound to map B used as a peer map from Map A. Thus the key reference resolves to topic `map-b/topics/topic-b-01.dita` as referenced from Map B.

This establishes the key-to-resource binding for the content *as authored*.

To generate a deliverable for Map A with apply the steps of the abstract process as follows:

Step 1: Determine peer dependencies

To determine the deliverable dependencies for Map A we examine Map A to find all peer map references. We find one, the one to Map B.

Step 2: Determine the specific deliverables required for each peer dependency

For this example we've decided to produce an HTML and PDF deliverable.

Step 3: Generate deliverable-specific maps for each deliverable dependency

We now process Map B to generate two deliverable-specific maps, one for HTML and one for PDF.

Map B HTML deliverable map (map-b-html.ditamap):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <title>Map B keydefs as delivered for deliverable "HTML"</title>
  <keydef
    keys="topic-b-root"
    scope="external"
    format="html"
    href="html/topics/topic-b-root.html">
    <topicmeta>
      <navtitle>Map B</navtitle>
    </topicmeta>
  </keydef>
  <keydef
    keys="topic-b-01"
    scope="external"
    format="html"
    href="html/topics/topic-b-01.html">
    <topicmeta>
      <navtitle>Topic B-01</navtitle>
    </topicmeta>
  </keydef>
  <keydef
    keys="topic-b-02"
    scope="external"
    format="html"
    href="html/topics/topic-b-02.html">
    <topicmeta>
      <navtitle>Topic B-02</navtitle>
    </topicmeta>
  </keydef>
</map>
```

Note that each of the key definitions from Map B is reflected here and that the topic titles have been captured as navigation titles. The non-key-defining topicrefs are omitted.

Map B PDF deliverable map (map-b-pdf.ditamap):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <title>Map B keydefs as delivered for deliverable "HTML"</title>
  <keydef
    keys="topic-b-root"
    format="pdf"
    scope="external"
    href="./map-b.pdf#page=1">
    <topicmeta>
      <navtitle>Map B</navtitle>
    </topicmeta>
  </keydef>
  <keydef
    keys="topic-b-01"
    format="pdf"
    scope="external"
    href="./map-b.pdf#page=2">
    <topicmeta>
      <navtitle>Topic B-01</navtitle>
    </topicmeta>
  </keydef>
  <keydef
    keys="topic-b-02"
    format="pdf"
    scope="external"
    href="./map-b.pdf#page=3">
    <topicmeta>
      <navtitle>Topic B-02</navtitle>
    </topicmeta>
  </keydef>
</map>
```

Note that each @href value includes a fragment identifier. These fragment identifiers are specific to delivery format, in this case PDF. The rules for fragment identifiers are defined by the format being accessed (e.g., PDF) or by the specific reader or viewer technology being used to access the deliverable. This example reflects the expectation that the PDFs for Map A and Map B will be delivered in the same directory, allowing the use of relative URLs.

Step 4: Generate the root deliverable map for Map A

To create this map we replace the original reference to Map B with local-scope map references to the Map B delivery-specific maps.

Map map-a-html.ditamap:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <title>Map A (HTML)</title>
    <!-- This map establishes the scope "map-a".

         The use case is that you want to be able to reference
         topics using keys of the form "map-a.foo" from outside
         this scope, but "foo" from within the scope, that is,
         topics that are only ever used from within this scope
         or for which "foo" will be defined in whatever scope
         the topic is used.

         We now include the map-a local keydefs within this
         scope so that these keydefs are within the scope
         and thus unqualified references to them will work.

         Note that we can't put the scope on the map with
         the keydefs because that map doesn't then contain
         the navigation topicrefs that we want to be in the
         scope.

         This means that each use of topic-a's.
      -->
  <topicgroup>
    <topicmeta>
      <navtitle>Keydefs Map A</navtitle>
    </topicmeta>
    <keydef
      keys="topic-a-root"
      href="topics/topic-a-root.dita"/>
    <keydef
      keys="topic-a-01"
      href="topics/topic-a-01.dita"/>
    <keydef
```

```
      keys="topic-a-02"
      href="topics/topic-a-02.dita"/>
  </topicgroup>
  <!-- Include Map B's deliverable maps as submaps
         and assign a keyscope to them. -->
  <topicgroup keyscope="map-b">
    <mapref
      href="../map-b/map-b-html.ditamap"
      scope="local"
      keyscope="map-b"
    />
    <mapref
      href="../map-b/map-b-pdf.ditamap"
      scope="local"
      keyscope="map-b"
    />
  </topicgroup>
  <!-- Navigation tree for map A: -->
  <topicref
    keyref="topic-a-root">
    <topicref
      keyref="topic-a-01"/>
    <topicref
      keyref="topic-a-02"/>
  </topicref>
</map>
```

Note that what was one map reference is now two map references, one for the HTML deliverable and one for the PDF. The HTML deliverable map is included first because this map is for the HTML deliverable of Map A and it follows that, by default, cross-deliverable references should be to the HTML deliverable for Map B. However, including the PDF deliverable map makes it easy to know that that deliverable exists and make it easy for authors to get it its key definitions in order to copy them into the Map A HTML deliverable map.

For example, if it was desired to have the link to Map B topic-b-01 link from the HTML to the PDF deliverable, the Map A HTML deliverable map could be modified to copy in the key definition from map-b-pdf.ditamap:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <title>Map A (HTML)</title>
    <!-- This map establishes the scope "map-a".

         The use case is that you want to be able to reference
         topics using keys of the form "map-a.foo" from outside
         this scope, but "foo" from within the scope, that is,
         topics that are only ever used from within this scope
         or for which "foo" will be defined in whatever scope
         the topic is used.

         We now include the map-a local keydefs within this
         scope so that these keydefs are within the scope
         and thus unqualified references to them will work.

         Note that we can't put the scope on the map with
         the keydefs because that map doesn't then contain
         the navigation topicrefs that we want to be in the
         scope.

         This means that each use of topic-a's.
      -->
  <topicgroup>
    <topicmeta>
      <navtitle>Keydefs Map A</navtitle>
    </topicmeta>
    <keydef
      keys="topic-a-root"
      href="topics/topic-a-root.dita"/>
    <keydef
      keys="topic-a-01"
      href="topics/topic-a-01.dita"/>
    <keydef
      keys="topic-a-02"
      href="topics/topic-a-02.dita"/>
  </topicgroup>
  <!-- Include Map B's deliverable maps as submaps
        and assign a keyscope to them. -->
  <topicgroup keyscope="map-b">
    <mapref
      href="../map-b/map-b-html.ditamap"
      scope="local"
      keyscope="map-b"
    />
    <mapref
      href="../map-b/map-b-pdf.ditamap"
      scope="local"
      keyscope="map-b"
    />
  </topicgroup>
  <!-- Navigation tree for map A: -->
  <topicref
    keyref="topic-a-root">
    <topicref
      keyref="topic-a-01"/>
    <topicref
      keyref="topic-a-02"/>
  </topicref>
  <keydef
    keys="map-b.topic-b-01"
    format="pdf"
    scope="external"
    href="./map-b.pdf#page=2">
    <topicmeta>
      <navtitle>Topic B-01</navtitle>
    </topicmeta>
  </keydef>
</map>
```

Note that the key definition has been added to the end of the map. This avoids having this key definition override any key definition that might have already occurred in the root map for the key named "map-b.topic-b-01". However, it would override any similar key defined in a submap included by Map A. By the rules of key definition precedence, a key definition anywhere within the root map document overrides any key definition for the same key name in any included map.

Note also that the key name is "map-b.topic-b-01". This reflects the fact that, when the key is in the context of the map-b-pdf map it is in the scope named "map-b" (because of the @keyscope on the <topicgroup> that contains the map reference). To get the binding we have to include the scope name as part of the key name when overriding the key within the root map, per the rules for scoped keys.

Step 6: Process the root deliverable map to produce the deliverable

Having performed step 5 and set up the key definitions for locations in the Map B deliverables, we can now process the Map A HTML deliverable map using a scope-aware DITA processor.

If we wanted to process the Map A HTML deliverable map with a DITA 1.2 non-scope-aware processor all that is required is to literally copy the scope names into the key names and references in the deliverable-specific maps. This is a relatively simple post-process on the map.

Performing this transformation on the Map B HTML deliverable map produces:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <title>Map B keydefs as delivered for deliverable "HTML"</title>
  <keydef
    keys="map-b.topic-b-root"
    format="pdf"
    scope="external"
    href="./map-b.pdf#page=1">
    <topicmeta>
      <navtitle>Map B</navtitle>
    </topicmeta>
  </keydef>
  <keydef
    keys="map-b.topic-b-01"
    format="pdf"
    scope="external"
    href="./map-b.pdf#page=2">
    <topicmeta>
      <navtitle>Topic B-01</navtitle>
    </topicmeta>
  </keydef>
  <keydef
    keys="map-b.topic-b-02"
    format="pdf"
    scope="external"
    href="./map-b.pdf#page=3">
    <topicmeta>
      <navtitle>Topic B-02</navtitle>
    </topicmeta>
  </keydef>
</map>
```

There is no change to the Map A HTML map because it does not itself define any key scopes other than the scope applied to map B.

Having completed Step 6 for Map A we have now produced an HTML deliverable for Map A that will correctly link to the Map B HTML and PDF versions once those versions are also delivered (which they may already be).