The Center for Information-Development Management
710 Kipling Street • Suite 400 • Denver, CO 80215 • 303/232-7586 • www.infomanagementcenter.com

ARTICLE REPRINT

From Monolithicity to Flexibility, Using DITA Keyrefs

Priscilla Buckley, SAP

mono•lith•ic
adjective \mä-nə-li-thik\

1. a: of, relating to, or resembling a monolith : huge, massive...
2. a: cast as a single piece <a *monolithic* concrete wall>
   b: formed or composed of material without joints or seams <a *monolithic* floor covering>
   c: consisting of or constituting a single unit
3. a: constituting a massive undifferentiated and often rigid whole <a *monolithic* society>
   b: exhibiting or characterized by often rigidly fixed uniformity <*monolithic* party unity>

Extract from <http://www.merriam-webster.com/dictionary/monolithic>

SAP is a leading provider of enterprise application software. Nearly 65 percent of the world's financial transaction revenue involves SAP software somewhere along the way. Traditionally, SAP's offerings such as SAP Customer Relationship Management or SAP ERP (SAP Enterprise Resource Planning) have not only been applications, but also end-to-end business suites: stable and reliable but at times also massive in scope and complexity, with major releases sometimes supported for twenty years and more. *Monolithic* may have been the word that came to mind when many people thought of SAP products.

New Product Strategy

A few years ago, SAP expanded its offerings in five market categories: applications, analytics, mobile, database and technology, and the cloud. SAP is accelerating change as the IT industry undergoes a structural shift toward investment in software-based innovation. Legacy products were broken up to be more nimble and release faster. Big data, cloud computing, mobility, and social networking present a significant market opportunity for SAP. Additionally, acquisitions complemented SAP's organic growth and boosted SAP in specific areas of strategic interest.

The challenges of SAP's new product strategy were—among other business units—evident in the Technology and Innovation Platform (TIP) unit, the unit that has been developing SAP NetWeaver, the platform for SAP's classic product portfolio. SAP NetWeaver documentation was as monolithic as the platform itself. The core documentation for a single release contained some 50,000 information objects, at delivery strung together and interlinked in a single massive HTML library. Five major SAP NetWeaver releases, with dozens of support packages, would eventually be supported concurrently, and proliferating code lines had created a byzantine evolution of content across the documentation set.

What a challenge for product documentation, then, when SAP NetWeaver's software components began to be decoupled, bundled in varying subsets within the larger platform, with asynchronous delivery cycles and, in some cases, shared components. The only way to deliver documentation was to componentize the documentation into sets mirroring the software components. If the linking binding them together could be reduced, they could be shuffled and re-packaged just like the SAP NetWeaver products.

Componentization

Componentization began with the definition of a basic architectural model to reflect the platform's software components, their hierarchy, and dependencies. SAP NetWeaver documentation was then segmented into dozens of building blocks, arranged in logical architectural layers from high-level products, through their components, and down through application servers to databases.

Information developers then spent months combing through their content, grouping topics into smaller subsets (one of which, nonetheless, contains 30,000 objects in two languages) and excising links that didn't reflect product dependencies, to render their content more manageable in changing contexts (see Figure 1 on page 2).

We Needed a New CMS

Componentization wasn't SAP NetWeaver's only challenge, however. Analytic applications from BusinessObjects, a business intelligence firm that SAP acquired in 2007, began to integrate with products from the SAP NetWeaver portfolio.

Shortly thereafter, a major upheaval emerged from within the TIP development unit: a revolutionary, in-memory platform called SAP HANA, which speeds up data transactions and analysis so radically, that in short order it became the foundation of SAP's technology strategy. From the start, it was delivered with SAP NetWeaver and SAP BusinessObjects components, but was slated to be associated with further SAP offerings.

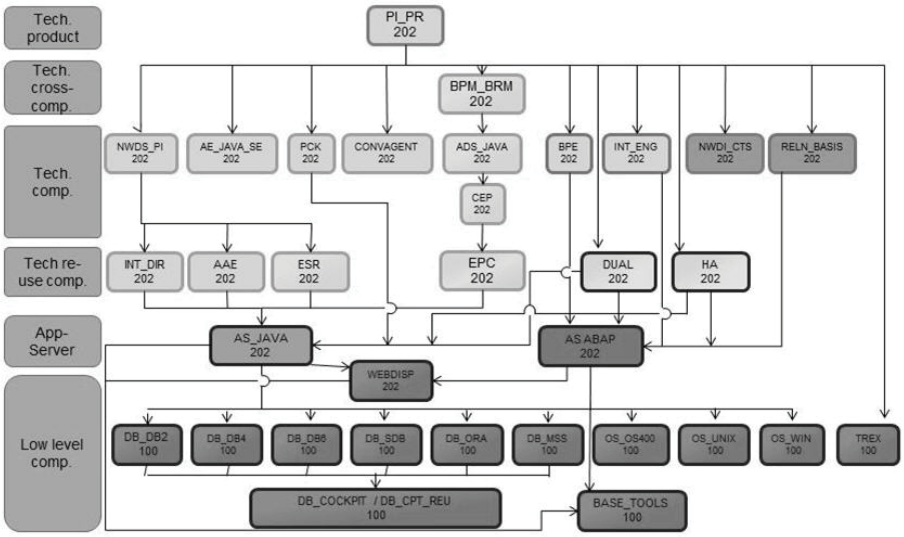

Figure 1: Example Content Model for an SAP NetWeaver Release

Within the TIP knowledge management unit (TIP KM), the once monolithic world of SAP had become a vast set of juggling balls, pins, and hoops, any combination of which might need to be kept up in the air at once. TIP KM was challenged to meet the product roadmap across two different environments and even more source file formats, with SAP NetWeaver in Microsoft Word and propriety XML in SAP's legacy content management system, and SAP Business Object's Darwin Information Typing Architecture (DITA) content in another legacy content management system (CMS), in which key components were shortly to be deprecated. The roadmap was only going to involve more integration and diversification. So in 2012, in an effort to optimize, align, and unify all product documentation into a single content management solution, we made the decision to consolidate across the TIP unit, on DITA 1.2, in a new third-party CMS.

Our primary requirements for the new CMS were these:

- Support for the type of content structuring model used in SAP NetWeaver componentization, to provide content building-blocks that could be assembled flexibly according to release requirements.
- Versioning capabilities to fit all release management strategies, without having to physically branch (copy) all content with each release. Given the massive quantities of topics we are managing, we didn't want obligatory multiplication of all files with each new release, even if many of them remained unchanged.
- Full compliance with DITA 1.2 for consolidation on a proven open-industry standard, with all of its benefits.

The TIP Content Architecture Model

In preparation for migrating TIP content into a single environment and to provide technical support for componentization, knowledge architects built on the SAP NetWeaver architectural model to create a generic model that could meaningfully organize content coming from anywhere in TIP, or even beyond.

This model contains thirteen layers:

- The bottom eight layers are the SAP NetWeaver model, dedicated to platform components.
- The top five layers are reserved for business applications either developed on top of the platform or without any back end. They run from a suite of end-user products at the top, down through applications, their components, and reuse components (see Figure 2).

The documentation content sets associated with each component in these layers are called *containers*. Containers are versioned, just as software components are versioned; a specific container version contains the version of the content maps/topics/images that are relevant for a specific release. Each container belongs to a specific architecture layer.

The hierarchy of all the containers required for a specific release is called a *dependency tree*.

The dependency tree also defines linking rules:

- Linking is only allowed between topics belonging to the containers in the dependency tree. This ensures that link targets are always relevant for the release.
- Topics in one container can only link to topics in a container in the same or lower layer in the dependency tree. This policy ensures that linking within the documentation set is logically sound and supports the need to re-bundle products without breaking links.

In Figure 2, Dependency tree 1 contains the content for a specific version of a suite of products, containing two products, one of which relies upon two platform products, and so on. Linking between components can only go in the direction of the arrows. For example, you can only link from a technical product in layer six to one of its components in layer four, not the other way around—logically speaking, the product relies on its components, not the other way around.

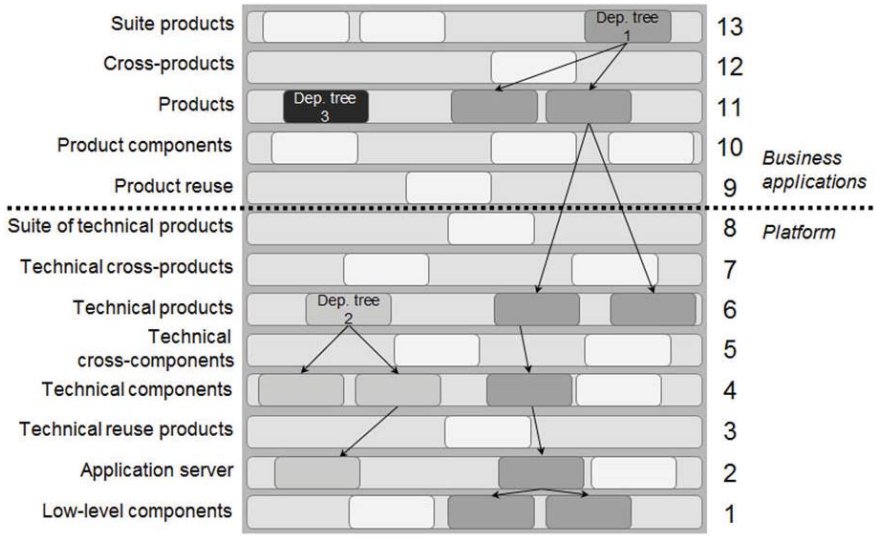

Figure 2: The TIP Content Structuring Model, with Architectural Layers, Content Containers, and Dependency Trees

This model is designed to work for releases of any scope, from sprawling and complex SAP NetWeaver releases, down to the standalone product SAP BusinessObjects Explorer, whose content is contained in a single container.

When we looked for a new content management system, support for this content architecture model was a priority.

Our Solution

The CMS evaluation resulted in our selection of Ixiasoft DITA CMS solution from Montreal-based Ixiasoft not the least because of Ixiasoft's elegant response to this requirement, using standard DITA. Ixiasoft's solution reposed upon extensive use of the DITA 1.2 keyref feature:

> DITA 1.2 introduces the "keyref" feature which provides an indirect addressing mechanism. You can use … keydef elements (new with DITA 1.2) to define keys in a DITA map. Topics can now be given a symbolic name (keys attribute) that points to a topic file path (href attribute). Future references to such topics are made using a key reference (keyref attribute). At a later point in time, if the topic is relocated, the path needs to be updated only in the map where it is defined. All other references will automatically pick up the new location.[1]

In our implementation, all references created between content objects would be indirect: keyrefs instead of hrefs and conkeyrefs instead of conrefs.

Containers

Containers are defined using specialized DITA maps (*container maps*) that use keydefs to reference all the objects (content maps, topics, and images) that "belong" to the container. Object IDs are used as each object's key. Each container is associated with an architectural layer, whose numerical value is inherited by all objects belonging to the container to support checking of compliance with linking rules.

At the beginning of a product release cycle, information architects define required containers or create new versions of existing ones, simply by cloning the container map. They then model the dependencies between the containers, establishing the dependency tree for the release. Dependencies between containers are created simply by dragging the child container and then dropping it into the parent container in the CMS interface. This creates a mapref from the parent container to the child.

The dependency tree in Figure 3 (on page 4) refers to the global set of content with potential relevance for the release.

Once a project's containers and dependency tree have been set up by information architects, authors are ready to begin their work. Shielded from the complexity of information architecture requirements, they can work in much the same way as authors working in a DITA environment using standard hrefs.

Content Maps

Content maps are specialized DITA maps that, like classic DITA maps, define the structure of topics, images, and submaps in a deliverable, but using keyrefs instead of hrefs. In addition, each content map references a single container map (to which it is said to belong); the keydefs in the container map, and those beneath it in the dependency tree, provide the resolution for the keyrefs in the content map.

References are resolved in real-time in the CMS environment and the oXygen editor so that as soon as an author adds a topic to a content map or a link to a topic target, titles are displayed instead of their keys.

1 Extract from "DITA 1.2 feature article: Keyref overview" by Sowmya Kannan (http://dita.xml.org/resource/keyref-overview-dita-12)

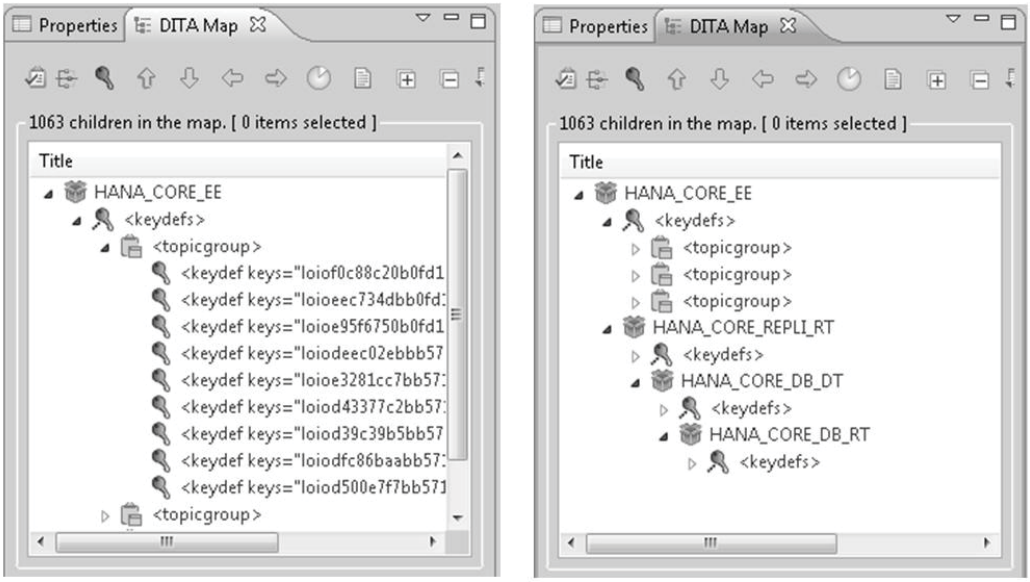

Figure 3: Two Views of the Same Root Container Map, with Keydef Keys (Left) and the Dependency Tree (Right)

The keydef/keyref mechanism also enforces the linking rules established by the information architect in the dependency tree (see Figure 4):

- When authors add topics to their deliverables or set links, any topicref pointing to a key that doesn't exist in the map's container or a lower container in the project's dependency

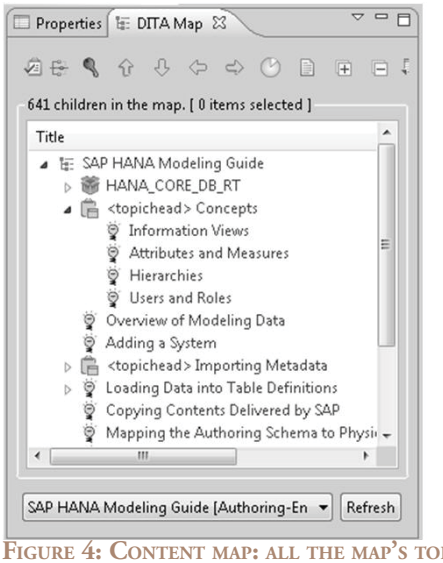

Figure 4: Content map: all the map's topics are referenced via keyrefs, whose resolution is provided by the keydefs in the container (HANA_CORE_DB_RT) displayed directly beneath the content map's title

tree, simply doesn't resolve, and is flagged by the CMS.

- Likewise, links are validated to ensure they respect the layer rules—a link can only point to objects at the same or lower level than itself.

Versioning

Each version of a container map references the versions of topics that are relevant for that container version. When you create a new version of a container, you create a new container map, with the same key definitions, pointing to the same content objects but without creating copies of the objects themselves, as in classical branching approaches.

Since content maps use only keyrefs to reference objects, they don't require manual adjustment to pull in updated objects. They rely on their container to provide the specific version of the referenced objects.

To create Version 2 of product documentation associated with a single container, Container_A_v1, an information architect clones the container (see Figure 5).

- A new physical version of the container file is created, Container_A_v2.
- All of the content maps in Container_A_v1 are also cloned and any references to them from other maps are adjusted, creating new versions of the maps that authors will use to update the documentation deliverables.
- By default, all the topic and image references in the container still point to the objects' versions in Container_A_v1.

When authors begin to work on the content for the new release, they simply open the cloned v2 content maps and work from there. The content maps automatically reference the new version of their associated container.

At this point, all topics and images are shared between both container versions. Authors can decide whether they want their modifications to apply to both versions or branch an object to make modifications in one version only. Branching

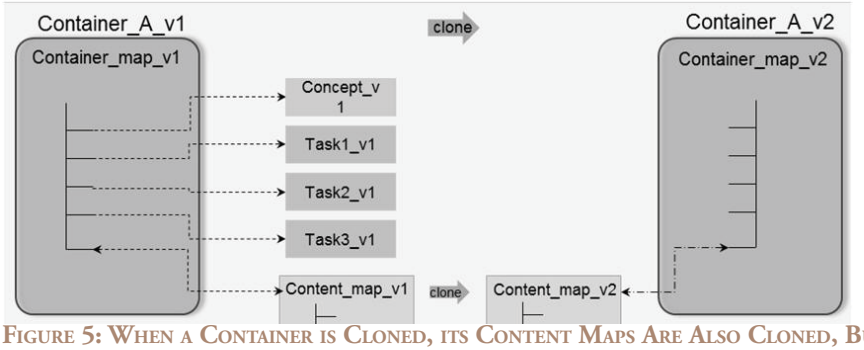

Figure 5: When a Container is Cloned, its Content Maps Are Also Cloned, But References to Content Objects Remain the Same

creates a new physical version of the object, with the same ID (key) but a different version. Its reference in Container A v2 is automatically updated. In this case, all content maps in Version 2 will automatically use this new file—without any action from the author—because the keyref they contain now points to it in the container as shown in Figure 6.

In this way, specific versions of containers are gradually updated with references to updated topics, but references to topics that haven't been changed remain the same. If you need to create new maps or topics for Version 2 of this product, you simply associate them with Container_A_v2 at creation and set their version from the beginning to v2.

Thanks to indirect linking, no existing links are broken when you move to the next version of product documentation and no manual updating is required. And thanks to the features designed by Ixiasoft to support this container and versioning model, authors manage versioning with ease.

Bundling

The advantages of indirect linking are also underscored when software components are removed from their traditional contexts and bundled with different product stacks.

The information architect determines the containers with relevant content for the new project. For containers whose content will be updated, a new container version is created. The information architect then models a new dependency tree using all the relevant containers, arranging them into a hierarchy according to the common architectural layering model.

When documenting the new release, authors either modify existing content maps (documentation deliverables) or create new maps associated with the updated container versions. Linking rules and resolution are defined by the new dependency tree. Any pre-existing references to topics that don't belong to the current dependency tree are automatically flagged as unresolved. And because we will largely limit linking to relationship tables, unresolved links can easily be fixed—or simply not rendered in output.

Conclusion

TIP KM released its first customer-facing product documentation in November, 2012, just eight months after our decision to go with Ixiasoft DITA CMS. The SAP NetWeaver documentation set is in the process of migration, and by the end of the year, the documentation content for a recent acquisition, Sybase, is intended to move into the new system. Already mostly in DITA, it fits right into our model.

We are DITA-compliant. We can version simply and elegantly, without redundancy. We can support the evolution of our products.

We are in a good place.

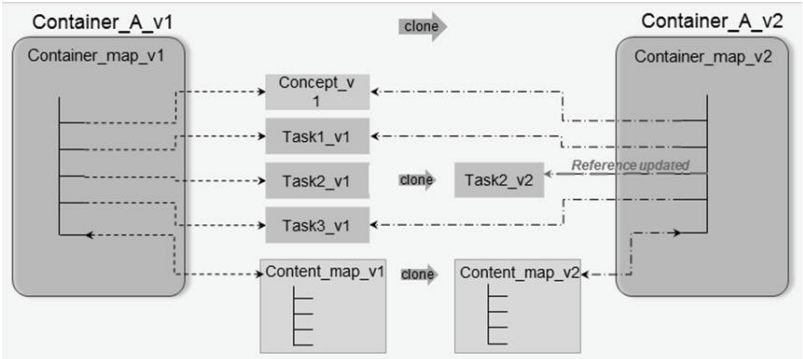

Figure 6: When An Author Decides to Make Changes in Topic Task2 v1 for Version 2 Only, the Topic is Cloned, and the References in Container Map v2 and Content Map v2 Are Automatically Updated to Point To It