

DITA—Four Letters You Need to Know

Introduction

DITA is one of the most important innovations in XML publishing in recent memory. And if you're using or plan to use XML for publishing technical documentation, you will encounter DITA sooner or later.

Short for “Darwin Information Typing Architecture,” DITA is an IBM invention that the company contributed to the community under the auspices of OASIS (<http://www.oasis-open.org>), the Organization for the Advancement of Structured Information Standards. (More information about the technical committee for DITA that OASIS formed can be found at <http://xml.coverpages.org/DITA-OASIS-CFP.html>.)

This document is not intended to be a “how to” guide for implementers, nor is it deeply technical. IBM has published a great deal of excellent, in-depth technical information around DITA at <http://www-106.ibm.com/developerworks/xml/library/x-dita1/>.

This author gratefully acknowledges the people and resources of IBM for his understanding of DITA, but he remains solely responsible for any errors, omissions or misinterpretations.

Table of Contents

Introduction	1
Why You Should Care About DITA	3
From Necessity to Invention—The Origins of DITA	3
Enter DITA	4
DITA Specialization—Matching Your Needs	5
Reusability—Myth or Reality?	5
Designing the Ideal Data Model	5
How Specialization Works	6
Easier Authoring, Better Comprehension	7
What’s in a Topic?	7
How to Categorize Your Information	7
Building Better Information	8
DITA Applied to Software, UI, and Formatting	9
Mastering Your Domain	9
Starting Out Small	9
Software Elements	10
UI & Programming Elements	10
Making DITA Work	10
You’ve Got DITA Content! Now What?	10
Tying it All Together—PTC’s Support for DITA	11
Summary	12

Why You Should Care About DITA

As with XML itself, DITA embodies a few simple constructs that have profound implications. Defining DITA is relatively easy; the hard part is describing its implications. So we'll begin with the definition.

DITA is an architecture based on XML for publishing technical information. In some ways, it is like DocBook, which was created some years ago as the basis for XML-based publishing of technical manuals, in particular for computer hardware and software documentation. (In researching this white paper, we came across an interesting set of resources at <http://www.namahn.com/resources/notes.htm>. Among several worthwhile white papers, they have a white paper on DITA that's more technical than this one and a little dated, but it includes a nice comparison of DITA and DocBook.)

There are two aspects of DITA that make it special:

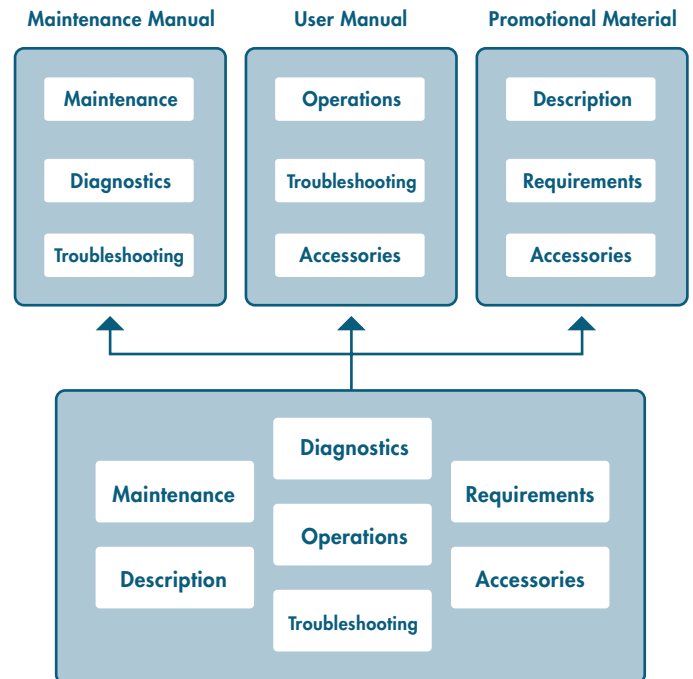
- **Modular**—DITA defines a Topic DTD that supports a modular approach to creating information. A topic is a component of information, not a complete document. A topic covers one aspect of a specific area of interest. (For example, this white paper could be divided into multiple topics: introduction, overview, origins, etc.)

DITA also defines a mechanism for combining topics into documents. This mechanism, called a “map,” also lets you define a hierarchy for the topics. For example, the map for a book could produce a hierarchy that consists of chapters, sections and subsections.

- **Adaptable**—The Topic DTD is similar to HTML in that it specifies a set of generic elements, where each element has different formatting such as titles, paragraphs, and lists. To adapt the Topic DTD to your specific needs, DITA defines a mechanism called “specialization” that allows you to define new tags that inherit their behavior and properties from tags in the Topic DTD.

Specialization allows downstream applications that are DITA-aware to handle an unknown tag by treating it as the tag from which it inherits its properties. For example, you could create a tag called “Procedure” that inherits from “Ordered List” and a tag called “Step” that inherits from “List Item.” Although you may want to add specific processing for Procedure and Step to your application, a DITA-aware application that knows nothing about these tags would handle Procedure and Step as if they were Ordered List and List Item instead. For instance, a DITA-aware publishing application that knows nothing about Step would format it as if it were a List Item.

Knowing the definition of DITA does not give you enough background for understanding its implications. So now let's start the story.



With DITA, you can create reusable modules of information that you can combine and assemble into different documents for a variety of purposes.

From Necessity to Invention—The Origins of DITA

Let's begin by re-visiting some of the key objectives of an XML publishing system:

- **Reuse**—To eliminate redundancy, improve accuracy, and reduce the effort to update information, XML helps you reuse and repurpose information so that you can create a single source of information in which a single change in the source will automatically update all of the documents where that information appears.
- **Sharing**—XML lets you construct your information in a way that allows other groups both within and outside your organization to incorporate your information automatically into their own processes, adding further value to the information you create.
- **Relevance**—You can use XML to help you create your information in modules that you automatically assemble according to the needs of each individual so that he gets everything he needs and only what he needs.

- Automation—To achieve these objectives cost-effectively, automation holds the key. XML makes that automation possible by allowing you to enforce the absolutely consistent structure that automated processes require.

The absolutely consistent structure that automation requires is defined by the “data model”—the DTD or Schema that prescribes which tags are allowed in your documents and how those tags may be used.

Now imagine you are a part of a huge company with an incredibly diverse product line and your responsibility is to oversee technical publishing for every single product. You have chosen to use XML because of its potential to achieve dramatic efficiencies in authoring, translation and publishing while enabling your organization to deliver more accurate, timely, and relevant information to your customers.

At the heart of your system is a data model you designed for publishing technical documentation by every group within the company. Your company-wide approach to publishing will not only reduce your costs for obtaining publishing tools and developing publishing applications, it will also allow your company to present a consistent face to your customers while combining content for various products into documentation that’s precisely tailored to each customer’s needs.

As each group starts up, however, they find holes in your data model. They have needs you did not anticipate and they want changes. Each change affects not only the data model itself, but also all the downstream applications that rely on that data model—the most notable of which are assembly and publishing. Each change may affect many publishing stylesheets as well as other processing logic, thus requiring updating, testing, documenting and re-deploying your application.

As more and more groups come online and ask you to make additional changes, and as existing groups find new opportunities that require further changes, you come to realize not only that the changes will never stop, but that you are falling behind—and will never catch up.

Enter DITA

Given these challenges, you decide a new approach is in order. You need to find a way to juggle several competing requirements:

- Serve the diverse needs of many areas of the organization
- Adapt quickly and easily to changes in business needs
- Support highly modular information to optimize reuse and personalization
- Keep things simple enough so that authors can become productive quickly

You begin by creating a DTD that contains all of the common formatting constructs that technical publications require. And what better

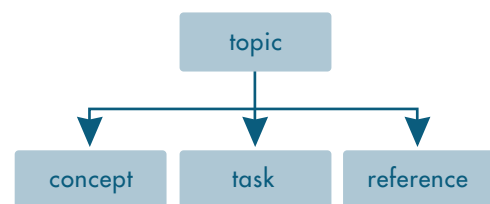
place to start than HTML, which has proven its flexibility across millions of Web pages? Titles, paragraphs, bulleted lists, numbered lists, italicized words—HTML represents all of these and more.

Your new DTD contains no tags specific to your business—just “generic” tags similar to those of HTML. These generic tags control the appearance of your information, but the tags represent almost nothing about the “semantics”—the meaning—of your information. Your new DTD also contains no document hierarchy—it represents only a module of information. (Later, you will come up with a way of combining modules into complete documents for delivery to your customers.)

You give your DTD a name—“Topic”—and then you design a mechanism so that you can easily adapt Topic to your specific needs. Inspired by work back in the 1990s on “architectural forms,” you create a method for defining new tags based on inheriting properties from existing tags. Any application that understands your syntax for defining new tags based on existing tags can process your new tags.

Because you are creating technical documentation, you then turn your attention to the specific types of information such documents require. Based on the principles of information architecture, around which exists a substantial body of research, understanding and experience that defines the most efficient and effective way to convey information to your audience, you define three specializations of Topic that will be the building blocks of your documents:

- Concept
- Task
- Reference



Concept, Task and Reference are specializations of Topic, where each is a different way to provide information about a topic. Although most DITA-aware systems should understand specifically how to process Concept, Task and Reference, any DITA-aware system can handle any specialization.

These specializations are not only your great idea, they represent another important innovation of DITA: using data models to guide authors to create information that’s both easier to write and easier to understand.

DITA Specialization—Matching Your Needs

“Specialization” is the mechanism within DITA that allows you to customize it for your own needs while maintaining the interoperability and reusability of both your existing content and your existing publishing system. Specialization also reduces the cost of maintaining your publishing applications as business needs change, because in many cases you can modify your data models without changing your downstream applications.

This section explains the impact and importance of specialization, but to appreciate specialization fully, we need to begin with understanding the principles of information reuse, which is one of the cornerstones of an XML publishing system.

Reusability—Myth or Reality?

Among the many promises of XML publishing, one of the most significant is the capability to reuse information in different ways. If you use XML, the story goes, you can insert the same information without modification in different documents and publish that information on different types of media, all while maintaining a single source so that one change can automatically update all the documents where that information appears.

This story is not a myth—many organizations have achieved this kind of reuse. The problems start when you try to reuse information in different types of documents, especially when the information comes from different sources both within and outside the organization.

Here’s the problem: to meet its own needs, each department creates a different data model (DTD or Schema) that contains tags specific to its purposes. For example, repair information could contain a <procedure> or <step> tag, while a parts catalog could contain <partnumber>, <description>, and <price> tags.

If you want to include repair information in a parts catalog or include parts information in a repair manual, then you have to change the data model to accommodate the new tags. Not only that, but you also have to change all of the applications that process your information. For example, you would have to change your stylesheets to accommodate the new tags so that you can publish information that contains the new tags.

This interoperability problem becomes worse as you try to reuse information across more document types, more departments and more enterprises. That’s where the promise of reusability becomes a myth, as the cost of accommodating different data models exceeds the benefit of incorporating them.

If each department limited itself to a simple, generic set of tags such as HTML, then information sharing would be easy. But using a simple, dumb data model would defeat many of the benefits of automation and content processability.

What a dilemma! The more an organization tunes its data model to meet its needs, the greater the value of their information. But the

more specialized the data model, the more difficult it is to reuse and share that information.

Despite this dilemma, many organizations have achieved significant benefits from content reuse and automation within a workgroup or department. Imagine the benefits that could be realized if those capabilities were to extend across departments and even across enterprises!

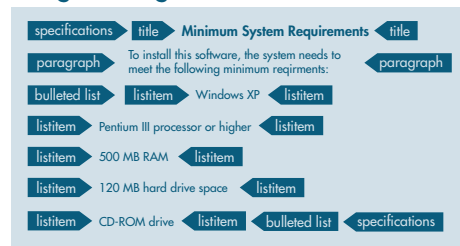
Designing the Ideal Data Model

The ideal data model would meet the needs of everyone in the whole world while remaining simple enough to learn. What would that data model look like?

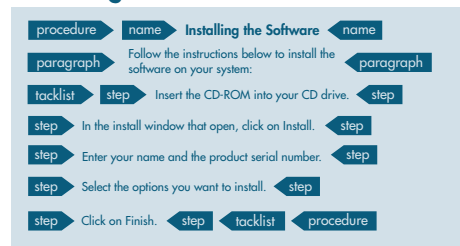
To meet everyone’s needs, the data model would have to contain hundreds of thousands of tags with tens of thousands of potential combinations. But to be simple enough for everyone to learn, it would have to be limited to fewer than 100 tags!

So that won’t work.

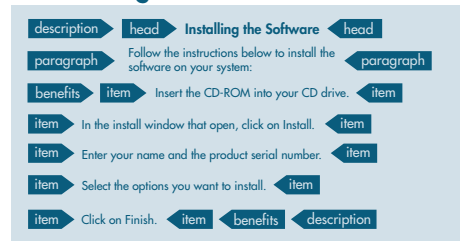
Engineering



Training



Marketing

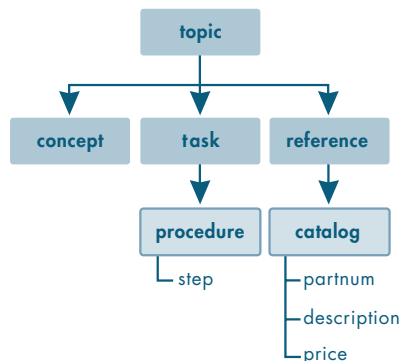


Each department has unique requirements for the content it creates. However, that information should be reusable across the entire organization. How do you create a data model that supports the needs of the entire organization while still remaining user-friendly for individual authors?

The designers of DITA decided upon an ingenious alternative: instead of a data model that tries to meet everyone's needs, they created a data model that's adaptable to everyone's needs. That was the inspiration behind "specialization," which is a feature of DITA that allows you to modify the DITA data model to suit your own needs while enabling other DITA-aware applications to incorporate your content even if they know nothing about your tags.

When you use DITA, you start with the base set of tags that DITA specifies. This base set of tags resembles HTML, which has proven its flexibility across millions of Web pages and several generations of improvements and provides most of the common formatting and structural tags that you would want.

To modify DITA to meet your needs, you add your own tags. If you did that to any other data model, adding your own tags would immediately inhibit or eliminate the interoperability of your content. But here's one of DITA's neat tricks: when you add a tag, you specify not only the name of the tag but also which of the base tags it most closely resembles. In other words, you create a new tag that inherits the properties and behaviors of an existing tag. (That's where the "D" of DITA comes in: it stands for Darwin for his work in biological inheritance.)



You can create specializations from specializations as this example illustrates, where procedure is a specialization of task and catalog is a specialization of reference. Within each specialization, you would create new tags to suit your specific requirements, where each tag is a specialization of a tag in topic or a specialization of topic.

For example, say you want to create a `<partnumber>` tag. Since part numbers probably appear inline, you would set it up to inherit from the `<emph>` ("emphasis") tag that's in the base DITA data model. You can still set up your application to process `<partnumber>` tags in a special way. For example, you could set up different formatting and customized validation against a database.

The benefit comes when someone else who knows nothing about `<partnumber>` tags can still incorporate and reuse your content without changing their underlying application. How? Because your data model will contain the inheritance information so that another DITA-aware application knows that if it does not understand `<partnumber>`, it should handle it like `<emph>` instead.

Here's one more example: say you want to set up a procedure that consists of a series of steps. You could set up the `<procedure>` tag to inherit from the `` ("ordered list," which is a numbered list) tag, and you could set up the `<step>` tag to inherit from the `` ("list item") tag.

How Specialization Works

Specialization complies with XML and XSL standards—there is nothing about DITA documents that makes them incompatible with these standards.

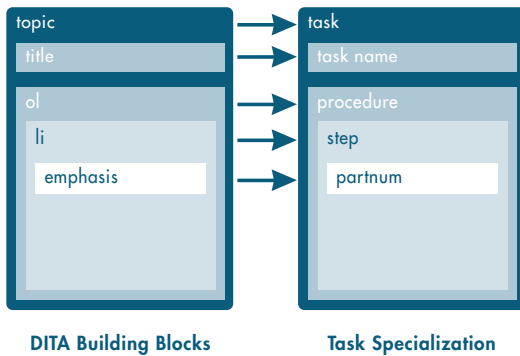
To adapt DITA to your own needs, you begin with Topic, the base DITA data model from which all specializations originate, and give it a suitable name. For an example, let's say you're creating a data model that you will call "Task." To create the Task data model, you start with the tags in Topic and then add your own tags. To continue our example, let's say you're adding `<procedure>` and `<step>` to your data model.

When you follow the DITA model of defining a new tag, you specify the tag from which it inherits its properties. This information is stored in a special attribute that the designers of DITA named "class." The class attribute contains the inheritance information. In a DTD, the syntax of the class attribute definition uses the ATTLIST keyword; the following example defines the inheritance information for the `<step>` tag:

```
<!ATTLIST step class CDATA "- topic/li task/step">
```

This ATTLIST declaration not only declares the "class" attribute for the `<step>` tag, it also specifies the default value for that attribute. To remain compliant with the DITA methodology, you would inhibit the author from changing this attribute value during editing.

In the example above, the class attribute for `<step>` is set to `"- topic/li task/step"` (the double quotation marks are not part of the attribute value). To DITA-aware applications, `"- topic/li task/step"` means that if the application knows about the Task data model, then it should handle `<step>` as a `<step>`. But if the application does not know about Task, then it should handle `<step>` as if it were `` within the Topic data model.



Presentation Tags	
Tag name	Description
	Bold
<i>	Italic
<u>	Underline
<tt>	Teletype (monospaced)
	Unordered list (bulleted list)

Specializations are based on the building blocks provided in DITA. If an application or stylesheet does not recognize the tags in the specialization, it handles them as if they were the DITA tags from which they originated.

To publish content based on DITA, you set up the stylesheet to match against the class attribute instead of the tag. The stylesheet for the base data model, Topic, can handle any specialized tag simply by examining the class attribute.

Easier Authoring, Better Comprehension

Now, let’s look in more detail at the three specializations that IBM has defined—concept, task and reference—and consider their impact on authoring and comprehension.

What’s in a Topic?

The basic building block of DITA is “Topic,” which is a short narrative with a specific purpose.

Topic consists of a basic set of tags for organizing and presenting information; some of the tags in Topic include:

Organizing Tags	
Tag name	Description
<title>	Heading or label; can be the main title (under <topic>) or a subordinate title (for example, under <section>)
<shortdesc>	Short description for link preview and searching
<section>	Division of topic; a topic may have multiple sections, but a section may not contain a section
<body>	Contains the main content of topic
<p>	Paragraph
	Unordered list (bulleted list)
	Ordered list (numbered list)
<table>	Table

Topic contains no tags that are specific to a specific industry, application or document type. Because DITA allows you to adapt it to meet your specific requirements, you would “specialize” Topic by adding your own tags.

When you add a new tag, you must also specify an existing tag which is most similar to your new tag (see the column entitled “Inherits From” in the next table). That’s how other DITA-aware applications can process your content even if they know nothing about your new tags—they will fall back to handle your new tags as if they were the tags from which they inherit (for example, from the next table, a DITA-aware application that knows nothing about <prereq> will handle it as if it were <section>).

How to Categorize Your Information

When you specialize Topic, you’re creating a new “Information Type” (also called “infotype”). For example, IBM created a specialization of Topic called Task which contains the steps to be performed to accomplish a procedure. In other words, Task is an Information Type and it contains specialized tags such as:

Task Tags		
Tag name	Description	Inherits From
<prereq>	Prerequisite: what the user needs to know or do before starting the task	<section>
<steps>	Series of steps that the user must follow to accomplish the task	
<result>	Expected outcome from performing the task	<section>

DITA allows you to create your own Information Types, which you can adapt from Topic or from another Information Type. For example, let's say you want to create a new Information Type called "Repair Procedure." You could start with Topic and adapt it for your needs, but you would find it easier to start with Task because it already has some tags you need.

To create a document, you assemble one or more topics or other Information Types. Documents can consist of any combination of Information Types. For example, you could have a single document that contained several Topics and Tasks as well as your own Information Types.

Building Better Information

Based on the principles of information architecture, around which exists a substantial body of research, understanding and experience, IBM has defined three Information Types—Concept, Task, Reference—specifically for technical documentation.

Other information architectures exist for technical documentation. For example, Information Mapping is a company that has developed a methodology and published research that shows remarkable improvements in both the act of writing technical documentation (measured in number of words and time to completion) as well as reading technical documentation (measured in time and comprehension). You can learn more about Information Mapping at <http://www.informationmapping.com/>. To see their published research on improving technical documentation, visit <http://www.informationmapping.com/collateral/IMI%20history%20and%20results.doc> (this link takes you to a Microsoft® Word document).

Regardless of the specific approach, each methodology shares common characteristics:

- Prescribes the types of information (for example: concept, task, reference) that authors can create
- Modularizes the information into small, reusable components
- Focuses the author on creating one component at a time
- Automates the publishing to free authors from the burden of design and formatting
- Quickly teaches readers to expect information categorized into specific types and presented in consistent ways
- Enables easy reuse of information components in multiple documents and easy sharing across multiple organizations

With only three basic information types for technical documentation, DITA keeps things simple. You can specialize any of these types for your specific industry and documents, but you should still end up with just three different information types in a single book.

The following is a description of each DITA information type:

- **Concept:** Provides an overview for Tasks or References. Concepts tend to be short and simple, in part because the author can only add other sections or examples after a section or example.
- **Task:** Contains the steps for the reader to perform for a procedure. There are many specialized tags in Task to support a wide variety of ways to present instructions as well as the information behind them.
- **Reference:** Describes the characteristics of a set of things such as a list of products, definitions or programming commands.

While each of these information types has certain special characteristics, they share many tags in common. In addition to organizational and presentation tags, which we saw earlier, there are also tags for linking (cross-references and related links), searching, navigation, keywords and more.

Intriguingly, DITA could lead you to calculate some metrics from your documents, such as:

- Average number of words in a component, which would help identify components that appear to have too few or too many words (you may find that different information types should have different target sizes)
- Ratio of information types that appear in a book, which would help you, for example, to identify when books over-emphasize reference material or under-emphasize concepts

In short, by applying the DITA methodology to technical documentation, you have a ready-made structure for your manuals. You can readily adapt to your requirements the three DITA Information Types—concept, task and reference. Your authors can master a relatively simple set of tags to become adept at authoring in XML, and they will be more productive not only because they are liberated from the responsibility for formatting but also because they are working within well-defined and field-proven templates. And most important, your readers will find the information they need more quickly and have to wade through fewer words and arbitrary organizational structures to get there.

DITA Applied to Software, UI, and Formatting

Domain Elements are specialized tags that IBM has defined for specific subject areas. They not only provide a well-conceived starting point for a specialization for subjects including programming, software, user interface and formatting, but also demonstrate how to go about applying DITA's support for specialization to whatever application you have in mind.

Mastering Your Domain

DITA creates a great starting point for a topic-oriented approach to creating information, but the key to making DITA work for your situation is to “specialize” it by adding tags that meet your specific requirements.

For example, if you publish service manuals, then you may want to create a tag called “procedure” that contains a list of tasks in sequence. In a DITA application, you would create `<procedure>` as a specialization of an existing tag. Since procedures look a lot like numbered lists, `<procedure>` would be a specialization of `` (ordered list), which usually appears as a numbered list.

While `<procedure>` could be used for many different types of service manuals, you may want to create additional tags that are specific to your industry or organization. For example, if you are a software company, you may have to document your application programming interface (API) and you may need several specialized tags for that purpose.

Specialized tags such as these are said to belong to a “domain,” which is just a fancy word for subject area. As part of the design of DITA, IBM defined several sets of specialized tags, which they call “Domain Elements,” to cover specific subject areas. These subject areas include programming, software, user interface and formatting. The following sections explain the purpose of each of these sets of Domain Elements.

Starting with the existing Domain Elements should help you reduce the time to develop your application while increasing the probability of being able to share your content with others who are in the same domain. After all, what sense does it make for you to create the `<parameterlist>` tag when other people use `<parml>` to mean the same thing?

Starting Out Small

Let's begin by examining the Domain Elements for formatting. These are called “typographic elements” and they exist for applications where you want to give the author control over the formatting of text.

But wait! If we have tags that purely control the appearance of an element, doesn't that violate the separation of content from formatting that XML encourages? Well, yes. And if you're opposed to violating that separation, then you can further specialize the Typographic elements to have greater meaning. For example, you could specialize the `<i>` tag, which specifies italic text, by creating tags such as `<booktitle>`, `<emph>`, `<foreignword>`, and so on, until you have specified every situation under which text should be italicized. Whether there's sufficient value in distinguishing among those different meanings of italics depends on your needs (and not on XML “purity”).

Your new tags, which you could call “semantic” tags because they're about the meaning of the text and not about its appearance, give you the flexibility to adjust their formatting later without re-tagging. That's certainly one of the virtues of XML.

But there may be situations where it simply isn't worth the effort to create a special tag to capture the meaning of your text, in which case Typographic elements are a perfectly serviceable fallback. We might not scold you if you decide to make the Typographic elements directly available to your authors—but then again, we might.

So let's take a look at the entire list:

Typographic Elements	
Tag name	Description
<code></code>	Bold
<code><i></code>	Italic
<code><u></code>	Underlined
<code><tt></code>	“Teletype”—selects a monospaced font like Courier, where every character has the same width; this may be useful when you want letters to line up vertically
<code><sup></code>	Superscript—makes the text smaller and places it higher than the surrounding text; useful for footnotes and mathematical expressions like $2^2 = 4$
<code><sub></code>	Subscript—makes the text smaller and places it lower than the surrounding text; useful for chemical formulas like H ₂ O

Software Elements

If you're trying to describe how a software program works, you will find the following Domain Elements useful:

Software Elements	
Tag name	Description
<msgph>	Message Phrase—contains a message that a program emits
<msgblock>	Message Block—contains a multi-line message or multiple messages
<msgnum>	Message Number—if any
<cmdname>	Command Name
<varname>	Variable—contains a variable that is input to a program
<filepath>	File Path—contains the name and optionally the location of a file
<userinput>	User Input—contains what the user inputs to a program
<systemoutput>	System Output—contains any kind of output from a program; (<msgph> is more specific than <systemoutput>)

There are rules to using these elements. For example, in the definition of these Domain Elements, varname may be contained in msgph, msgblock, filepath, userinput and systemoutput, but it may not be contained in filepath.

These structural rules are just as important as—perhaps even more important than—the tag names. It's relatively easy to convert one tag to another and back again, but it's much harder to convert one structure to another and back.

UI & Programming Elements

Rounding out the Domain Elements are those for User Interface and for Programming.

You would use the User Interface elements to describe in detail user input to a computer program. These elements include menu, keyboard shortcut, computer screen, and user interface controls such as button and menu item.

Unlike all of the other domains where there are just a handful of elements, the Programming domain has twenty-five elements because there are a lot of different parts to documenting how to write code in

a programming language. With elements for such things as keywords, code, variables and parameters, you can describe all of the details of a programming language.

Making DITA Work

By design, DITA-based content consists of chunks of information, not books. How do you turn these chunks into a coherent set of information such as an online help file or a printed publication? That's where maps come in.

And finally, where do you turn for tools to support building DITA applications, creating DITA content, assembling that content into coherent collections, and publishing that content to different media? That's where PTC comes in.

You've Got DITA Content! Now What?

One of the greatest virtues of the topic-oriented content that DITA promotes is that authors can write in relatively small chunks. Instead of facing a 1000-page writing assignment, authors work on information components that are only a few pages long.

To assemble those chunks into publications, DITA defines a “map” that lets you specify which chunks to include, the sequence of those chunks, and the hierarchical relationship of the chunks.

For example, let's say that you have written the following chunks:

- Topic: Carburetors
- Concept: How Carburetors Work
- Task: Tuning Your Carburetor
- Task: Replacing Your Carburetor
- Reference: Relationship of Carburetors to other Engine Components

To assemble those chunks into a publication, you would list them in a map. The map allows you not only to list which chunks to include but also to specify any hierarchical relationships.

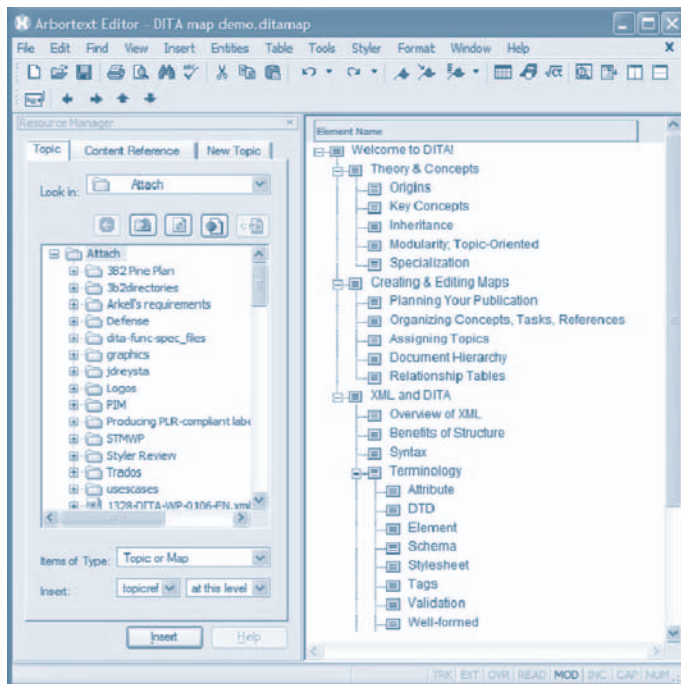
For example, in this case the last four chunks are subordinate to the first one, as the following hierarchy shows:

- Topic: Carburetors
 - Concept: How Carburetors Work
 - Task: Tuning Your Carburetor
 - Task: Replacing Your Carburetor
 - Reference: Relationship of Carburetors to other Engine Components

The primary element in a map is <topicref>, which acts as an inclusion mechanism (similar to file entities or XInclude) to specify a chunk to include in the current document.

The image below shows a map for our publication about carburetors. To establish the hierarchy, you can “nest” `<topicref>` within itself, which means that the `<topicref>` within is subordinate to the first one.

In the following example, the `<topicref>` for Carburetor contains the `<topicref>` elements for all of the remaining chunks, which makes those inside chunks subordinate to the outside chunk:



As part of the Arbortext 5.3 release, Arbortext Editor provides several tools to simplify the creation of topics (or specializations of topic such as Concept, Task and Reference) as well as "maps" that list the topics that are to be included in an information set (such as a book or an online help file) and how those topics will be navigated.

Tying it All Together—PTC's Support for DITA

Although the original intent of DITA was to comply with existing standards, it nonetheless introduced some features that require additional processing to support them. With the expectation that DITA will become important in a wide range of applications, PTC decided to invest significantly in adding DITA-specific functionality.

PTC has added to the Arbortext product line several features that provide or improve our support for DITA:

- **Authoring:** We have added several features to Arbortext Editor to deliver an easy-to-learn and highly productive environment for creating and editing topics and maps. These features include: drag-and-drop to add components to a Topic or to add Topics to a Map; spell checking and find/replace across all topics in a map; a new Resource Manager that puts reusable components at your fingertips; and a new Column View that makes a map look like a table of contents.
- **Assembly & Publishing:** Arbortext Publishing Engine uses your DITA map to assemble your documents and then publish those documents to multiple types of media, both print and electronic.
- **Specialization:** The Arbortext product line supports the specialization capability of DITA so that you can use a single stylesheet to control the style of Topic, Task, Concept, Reference and other specializations. Although XSL stylesheets already provide a mechanism to support specializations, we had to add support for specializations to our internal stylesheet format for editing.
- **Semantic tables:** The DITA DTD has some elements that should appear in the form of a table instead of narrative, so Arbortext products allow you to select tags that are displayed and printed as a table.

For example, you may want to create `<partnum>`, `<description>` and `<price>` tags for a catalog and have those tags display as a table. Without support for semantic tables, those tags would have to appear as narrative content:

`<partnum>` 123-456 `</partnum>`

`<description>` Carburetor `</description>`

`<price>` \$45.99 `</price>`

Sometimes information makes more sense when displayed as a table (as shown below) than as a list or a sentence.

To simplify editing, you can configure Arbortext Editor to display tags as table cells like this:

Part Number	Description	Price
123-456	Carburetor	\$45.99

- **Conref support:** DITA uses “conref” to include an information component within a document. This is a substitute for the file entity mechanism and XInclude and requires specific functionality that PTC has added in the Arbortext product line.
- **Topicref support:** DITA uses “topicref” to include a Topic (or specialization of Topic such as Concept, Task or Reference) within a map. This is a substitute for the file entity mechanism and XInclude and requires specific functionality that PTC has added in the Arbortext product line.
- **Doctypes:** Included in the Arbortext software are a set of doctypes for Topic, Concept, Task and Reference. A doctype includes the DTD, a stylesheet, and other customizations that are appropriate for that doctype.

Summary

One of the major benefits of an XML-based publishing system is the ability to reuse and share information across your enterprise. However, in most organizations, each division has its own documentation rules and requirements. Creating data models that are easy for authors to use but support the various needs of each division can be a daunting task. Managing such a data model as authors and corporate needs evolve and change can be nearly impossible.

DITA addresses these challenges of interoperability by providing a methodology in which unique data models can be created to meet the individual needs of each division in the organization. This limits the size of each data model making it easier for authors to use. However, because all of the data models are based on a single global data model, authors can still share and reuse information across the organization. This is because stylesheets and applications can process unrecognized tags in unique data models based on their global counterparts.

Authors are more productive because they can reuse existing information from any division in their organization. Updates to the source automatically update any instance of reuse. Therefore, authors spend less time recreating existing information and less time updating information that appears in multiple places. With reuse, updates are faster, easier and global resulting in information that is consistent and more up-to-date. Finally, ensuring that XML stylesheets and applications handle unknown tags based on the global DITA tag from which they were derived ensures that publishing can be automated without errors and updates to slow down the process.

In conclusion, DITA can help organizations realize the promise of XML; more productive authors, better quality information, and publishing processes that are not only faster but also less expensive.