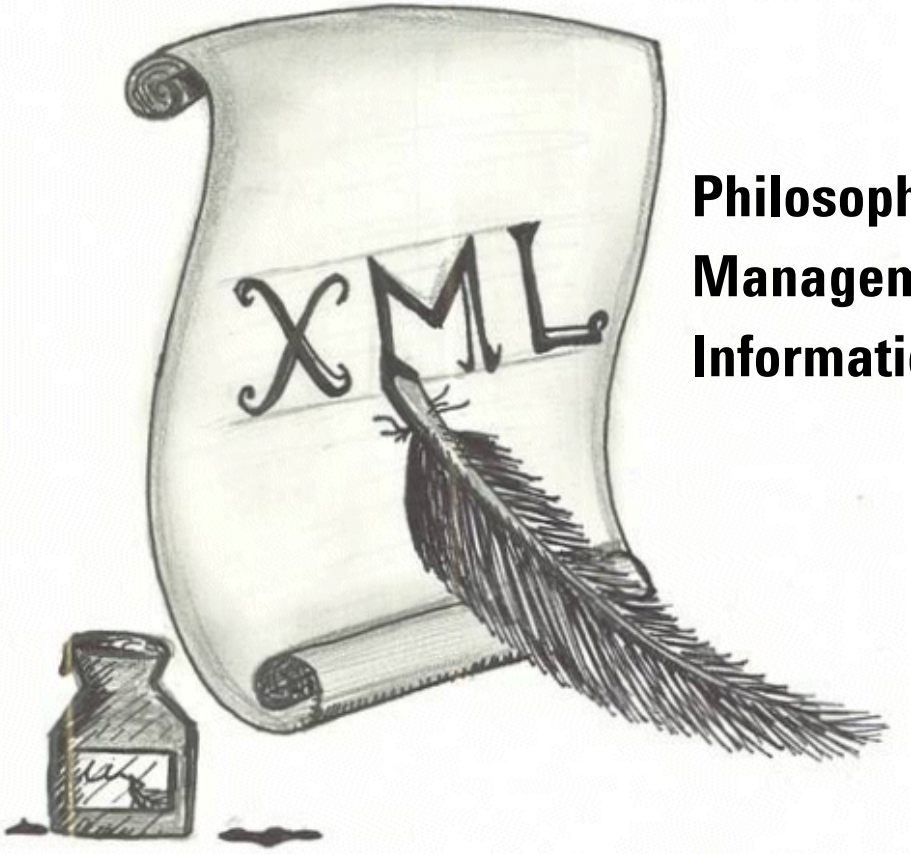


Philosophy of Content Management in User Information

The Art of Writing in a New Environment

Agilent Technologies

Notices

Manual Part Number

1234-56789

Edition

2nd Edition March 2010

Printed in Germany

Agilent Technologies
Hewlett-Packard-Strasse 8
76337 Waldbronn

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

For Research Use Only

Contents

Contents

1 Acknowledgment

This book could be written only because a lot of people have contributed to the implementation of the content management system . This is a good opportunity to say thank you to all.

First, there is the PHS user information team itself with Thomas Richwien as project lead. Thomas's professional management of the project enabled it to be successfully brought to this stage in a relatively short time-frame. Then there is Dr. Reinhard Zinburg, the technical lead and author of this book and there are Gabi Lichtenberger and Katja Bertz, who are working part time for our user information group and who contributed a lot with their user information expertise. I also want to say thank you to my daughter Ramona for the little art work on the title page.

A special thank you to Trevor Kemp, one of our external authors, who volunteered to review this book.

Then we have to say thank you to Dr. Stefan Bradenbrink (Reinisch AG), who accompanied us as a consultant during the project,and contributed his valuable practical experience in introducing content management systems.

Thank you also to Heide Ebert, the project lead at the Schema GmbH site. Her patience in listening to and understanding our special requirements made it possible for her and her team to implement and customize the system as it is right now.

We should also not forget to thank the PHS management, especially our supervisor Barbara Krull, for their understanding of our needs, that they finally approved the budget for the necessary investment.

1 Acknowledgment

Last but not least, thank you to all the authors who have already given, and who will give continuing feedback to enable us to make this system the perfect environment for their editorial work.

2 Preface

Agilent has always tried to communicate a strong, unified image to our customers and to the public at large. As part of Agilent's corporate identity, an Agilent Style Guide has been defined. It has its roots in the familiar Hewlett-Packard Writing Style Guide, a document that has helped to ensure the quality of HP literature for many years. It establishes standards for language use and spelling in all Agilent communications, and provides a reference for writers, editors, managers and all others who produce or review written content for Agilent.

Unfortunately, a lot of authors have not been aware of the exact content for whatever reasons, and the goal of the style guide has not always been achieved.

In 2007, the PHS user information group introduced a content management system (CMS) for generating user information. Beside its main goal of reducing costs for generating user information in various languages, as well as in the original English language itself, by reusing more and more content, it also has integrated Agilent's style guide to guarantee a better usage of its rules and advice.

But there are also new challenges for you as an author. The content is now separated from its structure, and especially from its format. In principle, this makes it easier to create new content, but it is different from what authors are used to, Furthermore, the content is based on xml-format, which changes the way you enter text. The next challenge is the reuse aspect. There has to be a distinction between content that is definitely unique and content that might be reused somewhere else. The content has to be written to be generic enough, so that it can be reused anywhere else. Last but not least, content is owned by an author, and therefore another author who is reusing this content cannot simply change it. Rules have to be considered to guarantee proper cooperation

in such a multi-user environment. This book will make the authors aware of these new aspects. It can also be seen as a short introduction to this content management system.

Both authors and those working in a user information department and toying with the idea of also using a content management system to support their work will gain a lot of knowledge and ideas from this book.

In this sense, enjoy reading this book.

3 From free style to structural writing with xml

An introduction to functional design and structural writing is given here.

From free style to structural writing

Functional Design

What does it mean?

The functional design has its origins in the accentuation of the spoken language. Through pronunciation and accentuation, a spoken text gains its functional meaning. We will explain this with a typical example. The phrase *"The door is open"* can have various meanings.

Its gains its special functional meaning as you pronounce and accentuate it:

- noticing: you just figure out that the door is open
- reproachful: you scold your son that he forgot to close the door after opening it
- appealing: you ask someone to enter the room through the open door
- angrily: you want someone to leave the room through the open door

In the written language, you can express these meanings by expressing the feelings using the appropriate words. In a technical documentation you can't do this. Here, the phrases should be short, and focus on the substantial. However, using layouting, you can denote a special functional meaning of a written text, for example using red color to express a warning. As we are going to separate the text from the layout, we have to find another way to assign a functional meaning to written text.

Jürgen Muthig (University of Karlsruhe – Technique and business science) and Robert Schäflein-Armbruster (University of Furtwangen). have developed a functional design that allows it to be applied to technical documentation.

In this functional design, information is organized hiearchically:

1. Top level: Information product (see "The Information Product" on page 17) (e.g. a manual)
2. Second level: Sequence of functions (e.g. how-to instructions)
3. Third level: The functional unit (e.g. a hazard)
4. Fourth level: Individual tagging (e.g. emphasizing)

Functional design works with modules (e.g. text modules). This simplifies reuse, standardization and automatization.

Example 1: Unstructured without functional design

To illustrate the functional design, here is a classic example. Let's take a look at the following instruction:

Set switch to position On. The green control lamp lights up. Ensure that all capillaries are connected properly to avoid spilling of liquid. Now the pump is ready for use. You can start pumping.

The instruction asks us to do something, but we should also be careful. Neither the action nor the hazard is clearly indicated. Why should we careful? What will happen if we are not careful enough? This is not fully explained. Therefore, let us enhance this instruction. Before doing so, we will analyze it .

This instruction consist of three functional parts:

- Action

  Set switch to position On.

  You can start pumping.

- Results

  The green control lamp lights up.

  Now the pump is ready for use.

- Hazard

  Ensure that all capillaries are connected properly to avoid spilling.

Example 2: Structured with functional design

Next, let's reorganize these parts, enhance the wording, standardize the warning and apply an appropriate layout to it.

The result is a structure that can easily be recognized as an instruction. You will be warned before you start doing something and it is also explained why you are warned and the possible consequences when the worst case happens.

To start pumping

WARNING

Liquid can be hazardous!

It may harm your health and environment.

➔ Make sure that all connections are tightened properly, that liquid cannot spill out.

1 Set the switch to the **On** position.
The green control lamp will light up.

2 Press **Start pump**.

Introduction to XML

The Extensible Markup Language (XML) is a general-purpose markup language It is classified as an extensible language because it allows its users to define their own tags. Its primary purpose is to facilitate the sharing of structured data across different information systems. It is used both to encode documents and to serialize data.

XML started as a simplified subset of the Standard Generalized Markup Language (SGML); it is designed to be relatively easily deciphered. By adding semantic constraints, application languages can be implemented in XML. These include XHTML, MathML, GraphML, Scalable Vector Graphics (SVG), MusicXML, and lots of others. Moreover, XML is sometimes used as the specification language for such application languages. XML is recommended by the World Wide Web Consortium. It is a fee-free open standard.

XML has found its way into the world of documentation. All the well-known software programs for editing such as Word or FrameMaker have incorporated XML as an additional format. Many content management systems have based their content and even their configuration on xml.

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities.

Further, it usually has three components:

- There is the XML-document itself. It holds the meaning of the information, the content (semantics).
- The structure of the XML document can be likened to a computer language grammar, which tells you how elements can stand in relation to each other. The grammar for a set of documents with the same structure is called a "document type definition" or DTD. DTDs can be very complex and large. They can also be based on XML and then they are called XML-Schemes.
- In order to present the document to a human reader, it needs to be formulated in some fashion. For that, the system needs a set of rules specifying how to render each xml-element: the "style sheet". This style sheet, called XSL (Extensible Style Language), is also based on XML. XSL offers more capabilities than the present Cascading Style Sheet specification for HTML, adding provisions for formatting elements based on their positions in the document, and handling of generated text.

In this way, XML can be transformed into other formats, XML, HTML, RTF etc. for which appropriate templates or style sheets may exist to present the content of the XML in a user-readable form. For HTML, this would be a CSS-file (Cascading Style Sheet) or for RTF, it would be a Word-DOT-file while FrameMaker-XML is uses its own DTD

Structural writing with xml

To put the functional design into an XML-context, an XML-scheme has been developed by the PHS user information team that helps authors of XML-documents to follow the recommendations of this design.

For the instruction as given in the previous example, this means that XML-elements are provided that put the structure of the given instruction into an XML-structure.

The XML-structure for this instruction is given as:

- Procedure

  It serves as container for the complete instruction.

- Goal

  This XML-element allows the goal of the procedure to be expressed (why the procedure is peformed).

  E.g. To start pumping

- Hazard

  Before taking any action it should be warned if necessary.

  Based on the ANSI-standard, a hazard consists of the cause for the warning or caution, the consequences of hazardous action and advice on how to avoid hazardous action.

- StepSequence

  The StepSequence-element encloses the steps of a procedure.

  - Step

    The Step comprises one action and its result in one XML-element.

    - Action

      The Action is the request of what actually has to be done in this step.

    - Result of action

      The result of an individual action visualizes or describes what happens when the action has been performed.

- Result of procedure

Last but not least, the expected result of the complete procedure is given to allow the actor to compare the expected result with the achieved result.

Our procedure example would therefore be tagged as following:

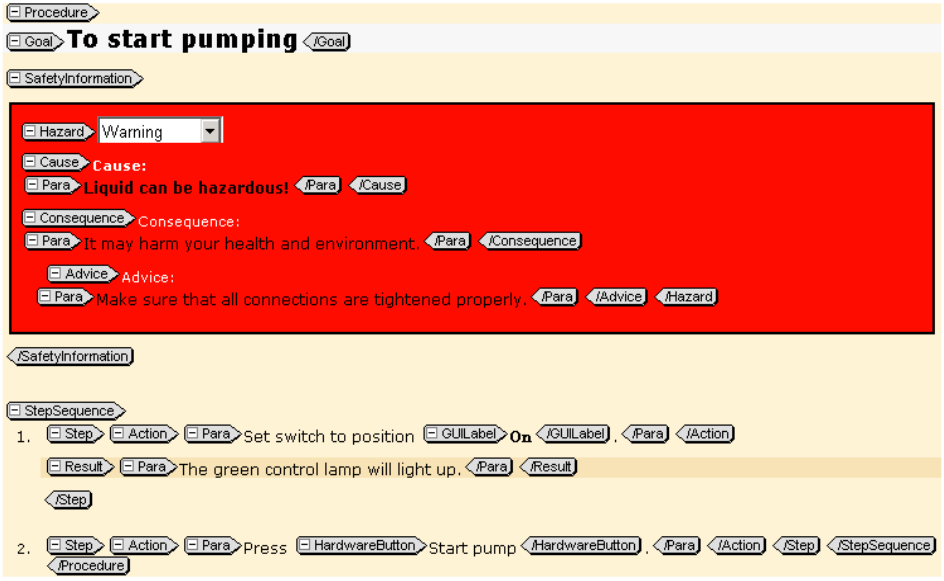


Figure 1 Procedure example in XML

The Information Product

An information product in its generic definition is any book, paper, map, machine-readable material, audiovisual production, or other documentary material, regardless of its physical form or characteristic.

In the context of PHS's user information, it is a specific XML-structure with content, which is used as the basis for making various documentation products, such as a manual (e.g. in PDF-format), an online help (e.g. HTML-Help) or a CD-ROM with HTML files etc. The atomic units of this information product are called i*nformation modules*.

We can say, therefore, that an *information product* is composed of *information modules*. ordered hiearchically to build a specific structure.

The information module

- defines the content of the information product
- builds up the hiearchical structure of the information product
- can reuse other information modules

4 XML Scheme

XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. in more detail. XML Schema was approved as a W3C Recommendation. The main parts of the XML-scheme as it has been defined by the PHS user information team are described with its most important elements: information product, reference and procedure. For a detailed description of the XML-scheme see the XML-scheme description manual.

XML-Scheme of Information Product

We have seen that an information product is composed of information modules. These can be references, procedures, steps, hazards etc. To distinguish between different information modules each has a title that you can set individually. The main structure element, which reuses other information modules such as references or procedures etc. is called a topic.

With this XML-Scheme definition of Agilent's user information, the information product is defined by the structure of topics, whose titles define the table of contents of the information product.

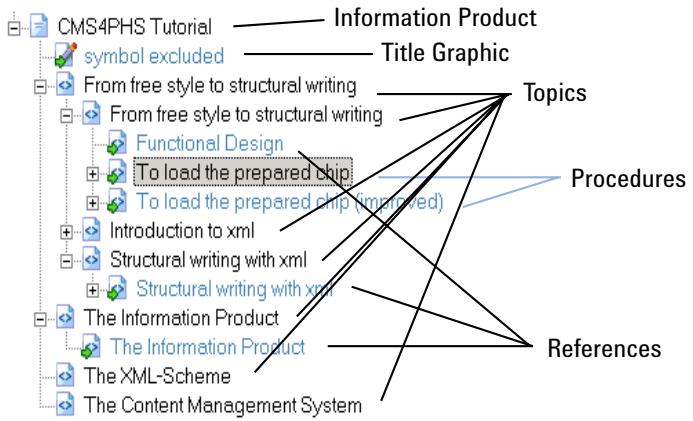
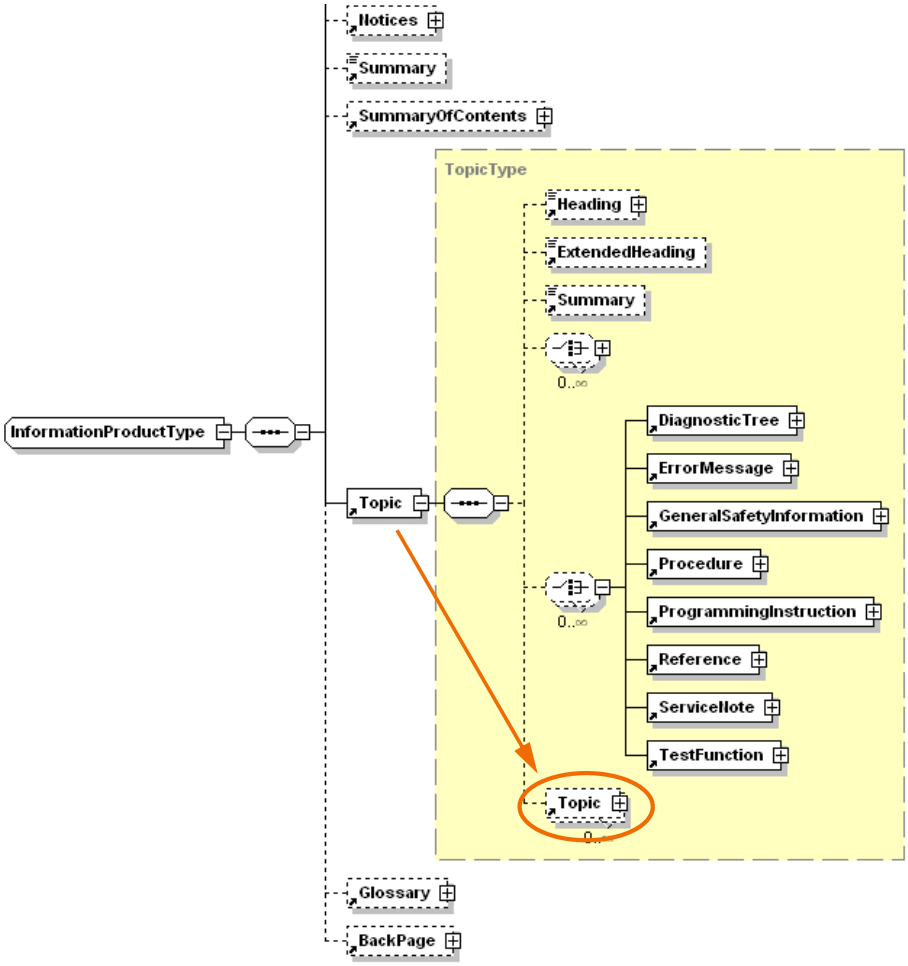


Figure 2 Information product topic structure

Once you have defined the topic structure, you can enter content either by writing new content (new references, new procedures etc.) or by reusing existing content. In Figure 2 on page 20 the reused information modules are visualized by folders, which are outlined in light blue color.

The following figure shows the complete XML-definition of the information product:



You can see that other information modules beside references and procedures can be embedded and reused within a topic. Furthermore, topics themselves can be embedded to create a more complex topic structure.

The first-level topics are embedded within the information product themselves. Just a reminder: in a manual, the first-level topics are called chapters and each usually also has a summary .

The information product starts with a title graphic and a title. Notices and a summary can follow. It ends with the back page.

NOTE XML-elements outlined with dotted rectangles are optional, while the others are required.

All xml-files that follow the rules of this xml-scheme of the information product can be imported to our cms database. This way, we have been able to migrate our existing FrameMaker documents into the database by firstly converting them to xml according to the xml-scheme of the information product. In principle, any document can be imported into our database, when it first gets converted to this specific xml format.

XML-Scheme of Reference

As you have seen, the topic can embed also other types of information module. Their schemes are defined separately but they are all part of the information product scheme. The scheme of the reference is one them that you will use most. Therefore it will be described here in more detail.

Reference information is defined as information to which you refer when you want to describe something, for example, a software user interface, a piece of hardware, a system, etc. To a user, this particular description serves as reference information. It can usually be described within a few structure elements such as paragraphs, lists or tables. Figures are also useful for illustration. Sometimes, it is necessary to substructure the given information into blocks.

The XML-scheme of the reference therefore considers all these structural elements:

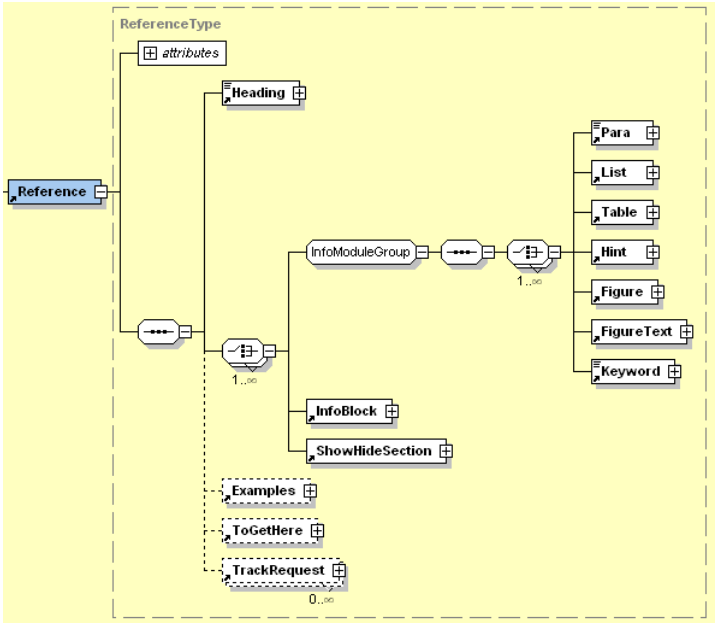


Figure 3 XML-scheme for reference

Each reference starts with a heading.

The most frequently used structural elements such as paragraphs, lists, tables, etc. are defined within the InfoModuleGroup. This means for you, that you can just use paragraphs, lists, tables etc. directly within the reference wherever you want. If a substructure is needed, the InfoBlock structure can be used:

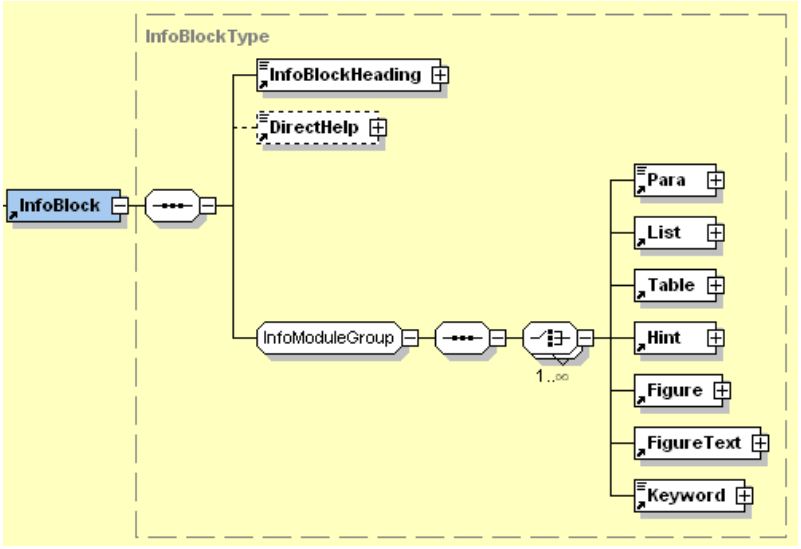


Figure 4 XML-scheme of InfoBlock

The information block itself consists of a heading and the same information module group as the reference, which allows the same structural elements to be used.

XML-Scheme of Procedure

The second most frequently used structure element that you will use is the procedure.You already have seen its principles in the first chapter (see “Structural writing with xml” on page 15). It describes actions that the reader of the information product must perform to fulfill a certain task. The basic elements of a procedure are their steps, which consists of the actions that must be performed.

The xml-scheme for the procedure is defined as follows:

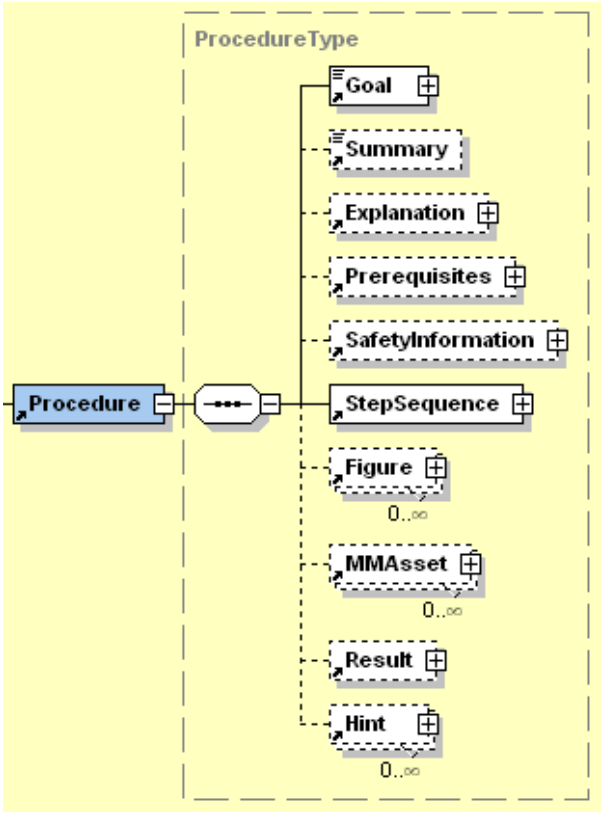


Figure 5 XML-scheme for procedure

The main elements here are the goal of he procedure usually followed by an explanation.

A list of the prerequisites may follow, where you describe the kind of tools, parts etc. that are needed or have to be prepared before starting the procedure.

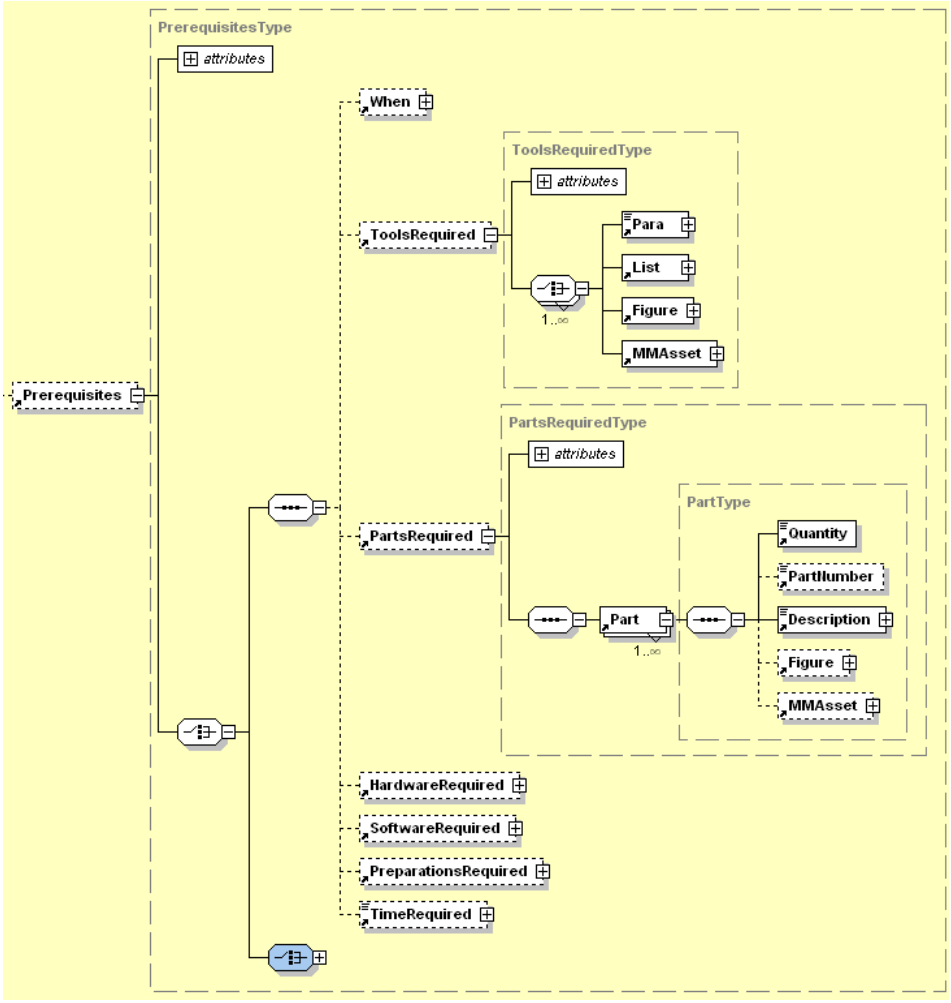


Figure 6 XML-scheme of the prerequisite element

In all of the prerequisite elements you can enter paragraphs, lists, figures and multimedia assets. Only the PartsRequired structure is a bit more strict. Here, you have to enter parts, which consist of its quantitiy, the part number and a description. For better illustration a figure and a multimedia asset can follow. Parts can also consist of (sub)parts.

Next, safety information can be given if the reader has to be warned against doing something wrong. Then the actions themselves follow, describing step by step what should be done to perform the procedure.

The steps are defined within the StepSequence element:

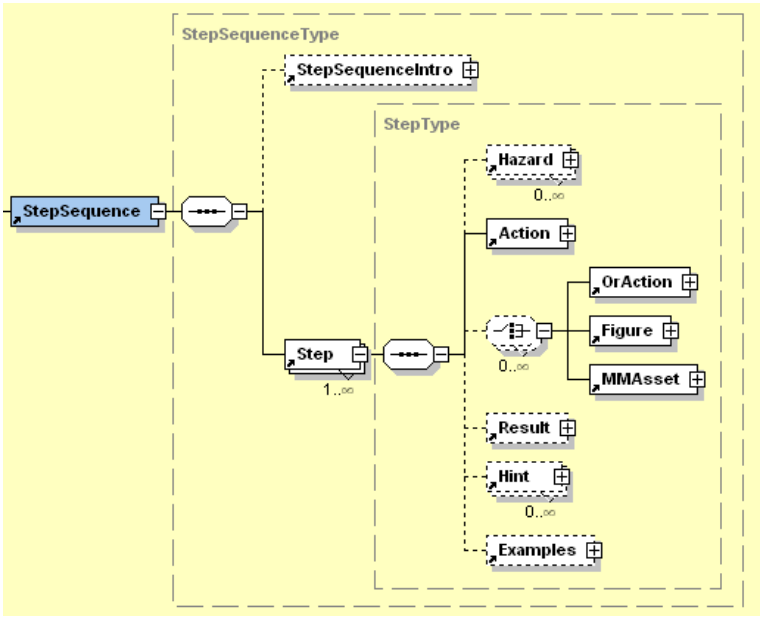


Figure 7 XML-scheme of the step sequence element

A single step may start with a hazard (warning or caution), which is specific for this step and therefore should not appear in front of the complete procedure. Then the description of the action is given followed by a description of the result that is observed after performing this action. The step can be illustrated with a figure. You also can give an example and tag it as such.

DocBook and DITA

Before proceeding to the usage of our xml-scheme, let's have a look at two other standard xml-schemes: the DocBook and DITA.

DocBook

DocBook is a markup language for technical documentation. It was originally intended for writing technical documents related to computer hardware and software but it can be used for any other sort of documentation. DocBook offers a large number of features that may be overwhelming to a new user. For those who want the convenience of DocBook without a large learning curve, a simplified DocBook was designed. It is a small subset of DocBook designed for single documents such as articles or white papers

DITA

When IBM developed DITA, it was primarily for the online documentation that was replacing traditional long printed user manuals, written in DocBook or IBM's proprietary IBMIDDoc. For the past couple of years, DocBook has remained strong in the area of system and software documentation, especially in the open source community, but DITA is expanding beyond online documents and now is taking aim at traditional book-form manuals. DITA V1.1 added xml-elements, which relate especially to the structure of manuals.

Conclusion

DITA V1.0 became public in May 2005 after we developed our own PHS user information xml-scheme. It is interesting that DITA has also called its structural elements "Topic" and that it uses a similar structure for procedures as we do. In other areas, DITA is very generic, where we have already defined the details. The current release of DITA V1.1 in August 2007 has already been extended by the addition of about 250 new xml-elements. For the V1.2 release, other more special elements will come and the hazards will also follow the ANSI-standard, which we have already implemented with our XML-scheme.

Overall, we can say that our xml-scheme is very similar to DITA; in some areas, it even goes beyond DITA, a situation that might change with future DITA releases. If it would be necessary to provide our information products in DITA-format, it would be quite easy to to so by applying a simple xsl-transformation to our information product.

4 XML Scheme

DocBook and DITA

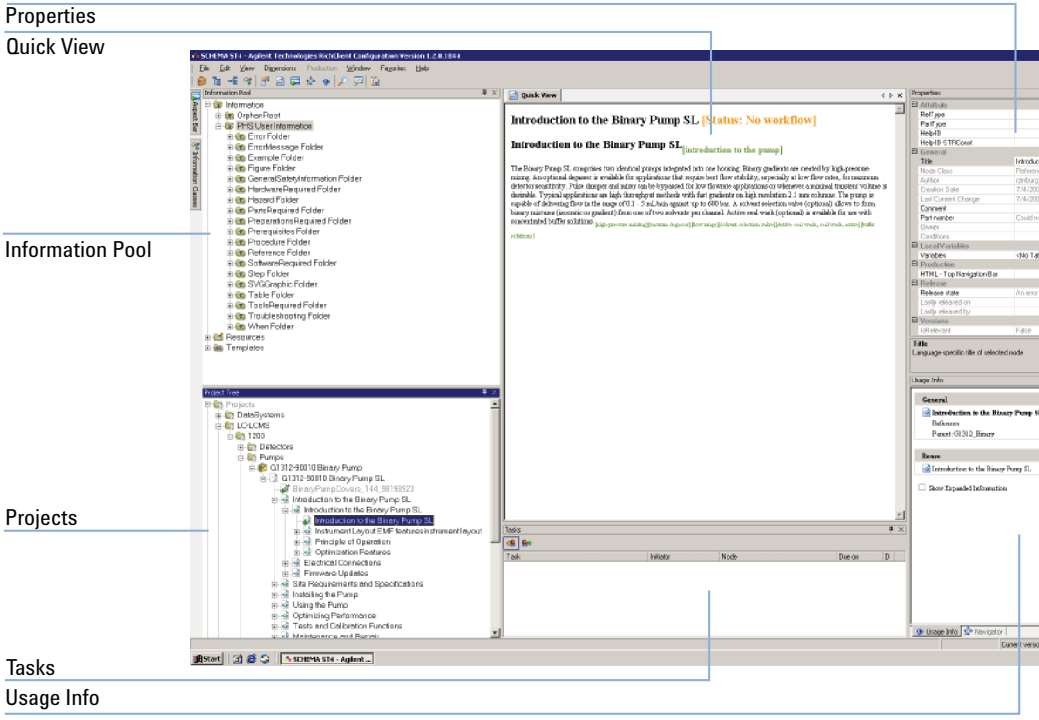
5 The Content Management System

The content management system as the PHS user information team has designed and implemented it is described . You will learn of what elements the user interface is built up and how the content can be modified. The editor, which allows you to enter and modify the content, is described.

The User Interface

The user information group uses two servers for its content management system (cms), one for the client and one for the application. To work with the cms, you first have to connect to the client, usually with a terminal program. Then you start and log into the ST4 Rich Client, which opens the user interface to the cms.

The graphical user interface of the cms consists of several sub-windows, called viewlets. They can be made visible using the **View** menu, and freely arranged in the main window. We recommend that you use these six viewlets and arrange them as given here as a start:



The Quick View Viewlet

The **Quick View** viewlet shows the content of an information module including the reference links to other information modules, and those that are embedded and - summarizing at the end - a link list to all linked and embedded nformation modules. The top line repeats the title of the information module (see “Title” on page 35).

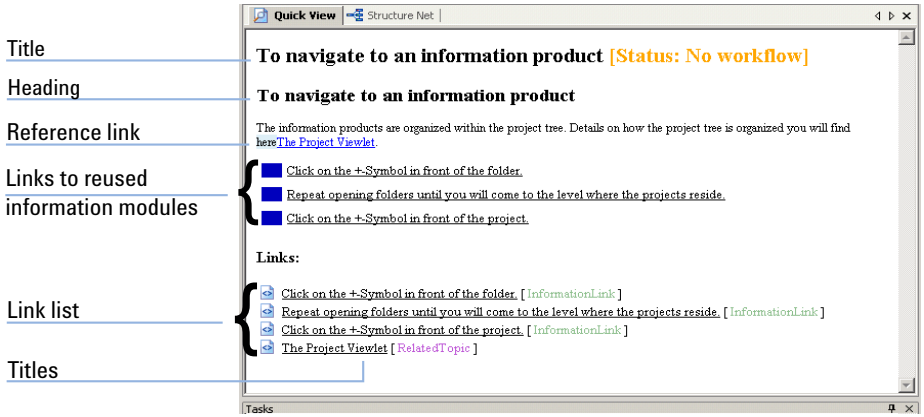


Figure 8 The quick view viewlet

The reference links are visualized as links like in an html page. They are also active as such, and you can navigate to the appropriate information module to see its content by clicking on this link as you would do on an html page.

The link to the embedded reused information module is visualized by a blue square, with its name on the right side. This link is also active, and you can navigate to its content by clicking on it.

Actually, the quick view shows the xml based content as html using a special XSLT style sheet to give you the impression of WYSIWYG.

The Project Viewlet

The main viewlet for you is called **Project Tree**. It contains the information products, and combines the structure of a file system and that of a table of contents in a common tree view.

First, you navigate to the information product as you do in a file system, and once you have reached the information product itself, you navigate further to the topic (chapter or section) of it as you would in a table of contents.

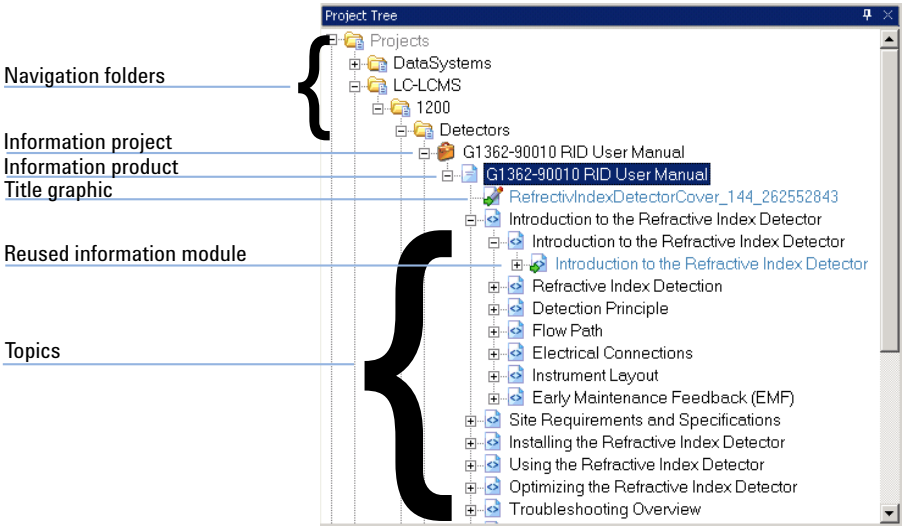


Figure 9 The project tree viewlet

Overall, in this tree view you distinguish between navigation folders, the project folder, the information product folder itself and the topics that build the structure of the information product, and last but not least the reused information modules. These titles represent the table of contents of the information product.

The Topic

The topic is the most important structural element, which you should understand very well. As already, mentioned the topic builds up the structure of any information product. It can itself have content, but it usually consists only of a heading and another embedded reused information module. The embedded information module is visualized as a subfolder of the topic outlined with light blue color in the project viewlet, and as a blue square along with its title in the quick view viewlet.

The topic itself resides in the information pool, which will be described in the next section (see “The Information Pool” on page 36). Take a closer look at the contents of the topics here. They mostly have a heading, sometimes an

introduction part and nearly always a main part You can easily distinguish between these items by looking at the topic content in the quick view. The introduction is the individual part and is written directly into the topic. The main part is written in the embedded information module (blue square) and therefore can be reused in other information products. Reused information modules must be translated only once, which saves additional localization costs. It also ensures consistency of presentation of an information product in different documents. This is the basic idea of reuse.

Titles and Headings

At first, you might be puzzled by all the titles and headings that you see in the tree view and in the content (quick view). They all serve a special purpose.

Title

The title of a topic or reused information module reflects the name within the tree view of the project viewlet as well as of the information pool viewlet (see "The Information Pool" on page 36). It is also shown in the preview as the top title (see "The Quick View Viewlet" on page 33). This name is *not* part of the content and therefore will never be translated. This has the advantage that you are able to read the titles (table of content) even you are looking at a localized information product e.g. a Chinese one. Since this title is independent of the content, it may differ from the heading in the content.

Heading

While a topic *can have* a heading, each embedded information module *must have* a heading. In the procedure, the heading is called goal to better serve its purpose there.

When the information product of the CMS is turned into a final document (e.g. a manual), the individual topic heading overwrites the generic heading of the embedded information module. This way, you can individualize the headings of the reused generic information modules. If the topic heading is not given, the heading of the reused information module is shown.

What happens when multiple information modules are embedded? In this case, the headings of the embedded information modules become subheadings. In an html production, they all will become part of the same html page given by this topic. However, stick to the rule to build up the structure only out of

topics in which only one information module is embedded. This way you also get individual html pages in an html production. The structure of a manual relies basically on this rule.

Variants

We have seen that each project can contain one information product. This is the typical situation. However, there is an exception. It is possible to create one or more variants of the information product, its topics and even of its reused information modules. A variant is a copy of the original and allows its modification, even when the original is reused and has a different owner.

A variant information product can be a mix of topics and reused information modules, whereby some are real variants or copies and others are still 100% reused. Any of the reused information modules in the variant information product can be also turned into a variant if there is a need for it.

Typically, you can create a variant when the resulting information product is very similar in structure and content. For example, our 1200 user manuals are a subset of their related service manuals and therefore their information products are defined as variants of the service information products.

As already mentioned, this type of variant consists of a copy of the original at the level of an information module. There are also variants possible where you don't have to create a copy of the information module. They are based on variables an will be described in the section “Creating Variants with Variables” on page 46.

The Information Pool

The next important viewlet is that of the **Information Pool**. Here, e.g. in the folder for PHS User Information, you will find all reusable information modules that you can embed in an information product. They are ordered by their functions e.g. procedures, steps, references, hazards etc. The hierarchical order underneath is mostly the same as it is for the position of an information product in the project tree view.

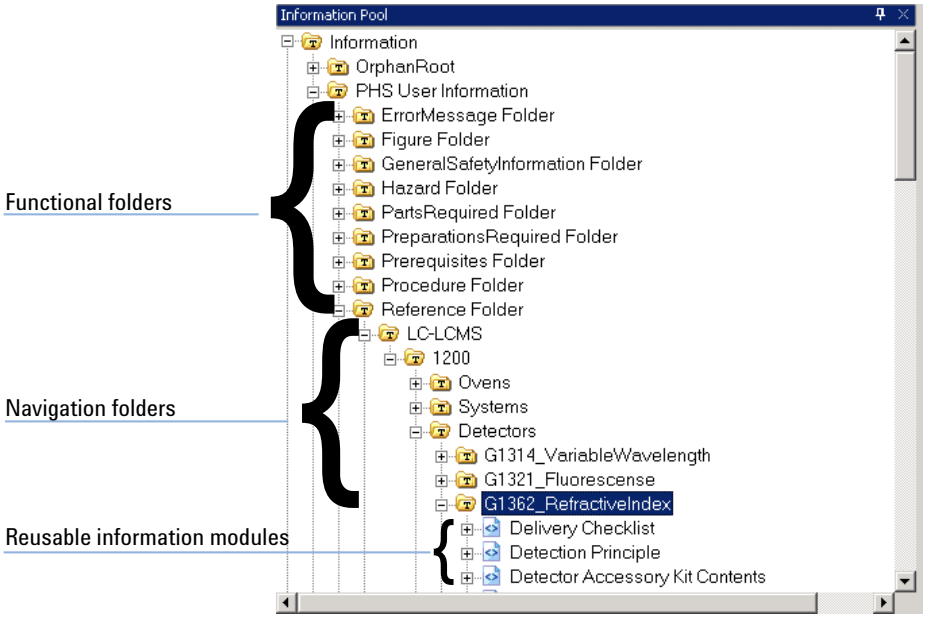


Figure 10 The information pool viewlet

In the resources folder, you will find all the graphics, videos or audios as they exist so far, in the same hiearchical order. Last but not least, in the templates folder you will find some predefined structures for information products. Some of these information modules themselves have other information modules embedded from the pool, such as the procedure, which reuses steps, both being reusable information modules.

To find the individual information modules, you navigate through the same structure as you navigate to find an information product in the projects viewlet. Since mostly similar information products embed similar information modules, this structure helps you to easily find the appropriate information module for reuse.

The Usage Info Viewlet

The **Usage Info** viewlet informs you where an information module has been reused.You can click on its name (it's an active link) and you will navigate to the information module (child), which has this information module (parent) embedded e.g. from a reference (child), which has been embedded in a topic (parent). To close the circle you can right mouse click on the reference underneath the topic in the project tree view and select the menu item **Select origin** from the context menu. It will lead you back to its position in the information pool.

The Property Viewlet

The **Property** viewlet contains settings that are assigned to each information module such as the type (class) of the information module (e.g. topic, reference, procedure etc.), its author and owner, its creation or modifcation date etc. There are a few properties that you, as an author, can set and that are therefore described below

For the information product and for the topic, you will find a set of variables. Here, you can set the value of one or more variables, which allows you to use it for conditional texting or within a paragraph as variable text. For example, you use the variable ProductName for defining the name of the product that you are going to describe in this information product. Then you can use this variable in the content text instead of repeating the complete name each time. Or you define the DocumentType of a particular information product and in the embedded information module you have a condition specifying that a paragraph should be used only when this information module becomes part of an information product of this particular document type. The setting of the variable is valid for the topic or information product, where it has been defined and for all topics underneath in the structure, unless it gets overridden by another value.

There are two more properties that can be set only for an information product, and you should be aware of them. They are called ProductPath and ProjectPath. They define the structural path (navigation folders) of their reused information modules in the information pool as described above (see Figure 10 on page 37). This becomes very important when you create new

information modules during editing. These variables tell the system where to store the new information modules in the information pool when checking the content into the database.

Now you might ask, what about properties of the information product such as the product number, copyright date etc. which you might be familiar with from FrameMaker? They are part of the content of an information product and therefore are set when editing the content of the information product module itself in the editor.

A special role is played by the product number. It *has to be* set in the content of the information product module (see "The Information Product" on page 17) and it *can be* additionally set as variable to be used as such in the content as variable text, or more likely for conditional texting to allow, for example, paragraphs to be part of a specific information product only when this paragraph is related to the given product number.

The Task Viewlet

In the **task** viewlet, you are informed about self-initiated and incoming workflows. Incoming workflows can be followed up by accepting or refusing them using the context menu (right mouse click on the workflow task).

Currently implemented workflows are

- *Ownership request workflow*

  Ask for ownership of information modules that you do not own in order to do modifications .

- *Change request workflow*

  Ask the owner of information modules that you do not own to do modifications.

- *Review workflow*

  Start the review process to prepare the information prodcut for release.

- *Find suspect links workflow*

  What are suspect links? Assume that you are reusing an information module that you do not own. From a physical point of view, this information module is not part of your information product. You just link to it and only when you produce the final document does it become part of it. When the owner of this information module changes its content based on a released

version, a new version of the information module is created. For consistency reasons, your information product still refers to the old version, to ensure that your content is not changed automatically, but your link becomes suspect to indicate this change of version. This workflow informs you to check this suspect link and update your reuse to the latest version if appropriate.

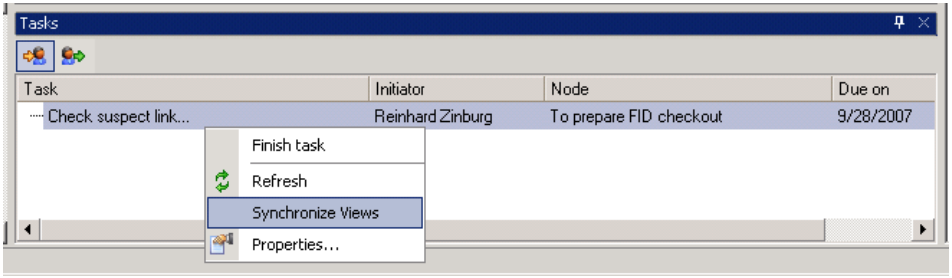


Figure 11 Task: Suspect link

The Editor

The editor is the main tool where you enter or modify the content. It's an editor (XMetaL) that was designed especially for editing xml-based content. You start the editor by double clicking on any information module in the project or in the information pool. The content of this particular information module and all the content of its substructure is checked out of the database and loaded into the editor. We recommend that you keep these chunks small enough not to lose the performance of the editor. Also, the checkout and checkin processes need time to prepare all the content for editing. An exact checkout/checkin time cannot be given, and can also vary, because the performance is also dependent on the number of authors who are working in the system in parallel.

Editing the First Time

When you open this editor to edit content the first time, you face two aspects of this editor compared to 'normal' text editors that you might not be used to, :

- The number of windows within the editor (viewlets)
- The number of views for editing the text
  - the text view
  - the normal view
  - the tags-on view

The Editor's Viewlets

The View menu allows you to switch certain views on and off, which, when visible, can be freely arranged on the user interface.

We recommend that you switch on the following views only:

- *Editing panel*

  The area where you are editing the content
- *Insert element viewlet*

  The list of xml-elements available for proper tagging of the content

- *Resource viewlet*

  The help panel with the integrated Agilent style guide

We recommend that you arrange them as given in the example below:

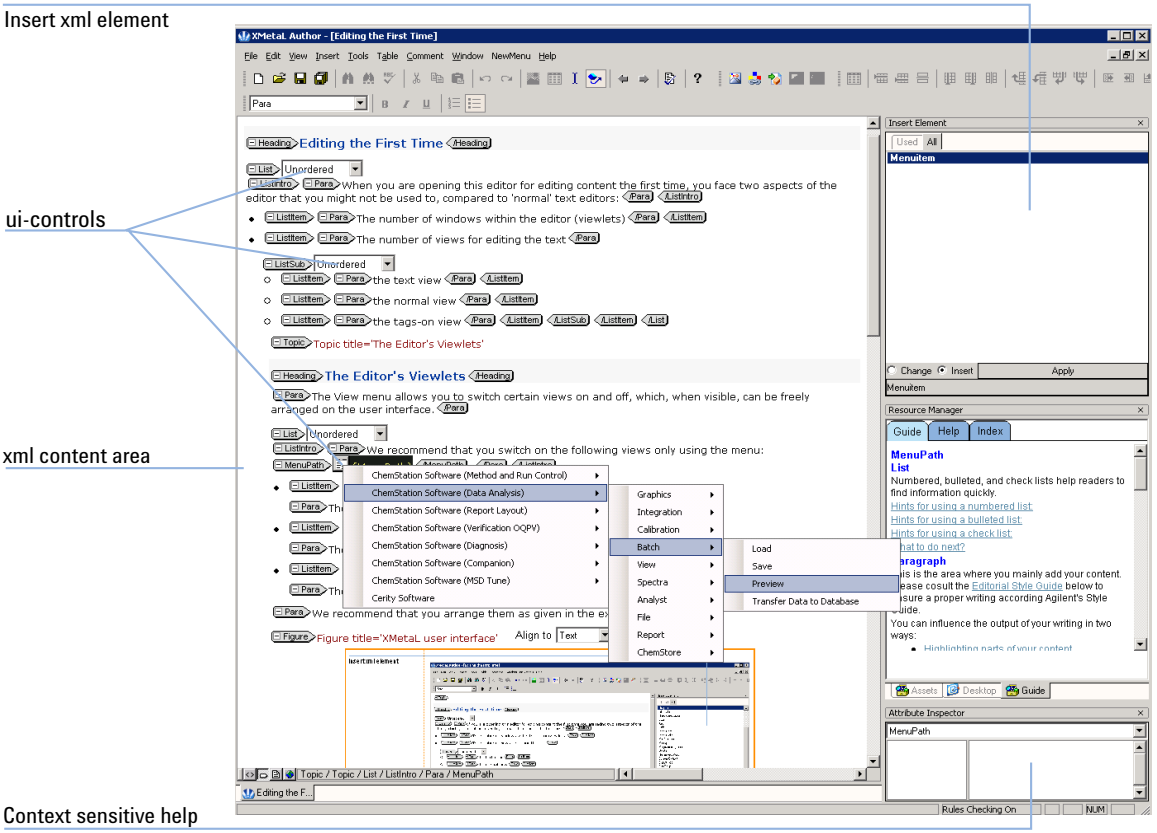


Figure 12 User interface of XMetaL

The Text View Modes

The editor offers three modes of viewing the content.

The Normal View

The **Normal View** displays the content without showing the xml-tags at all in a kind of WYSIWYG. The WYSIWYG is not identical to what you get in the final information product. The final information product can be a manual but also, for example, online help, and this same content might look different in each output format. The focus of this layout is to distinguish easily between the various types of functional xml-tags and to increase its recognizability, for example, that of a hazard.

You will find some ui-controls (user interface controls) embedded, which allow you either to enter directly some tag attributes with specific predefined values, or to modify structural aspects such as switching between a bulleted or an ordered list. There are also buttons that allow you to enter text in a more elegant and faster way. For example, with the menu button, you can enter a complete correctly tagged menu selection by using a menu (see example in Figure 12 on page 42)

You might want to use this view for editing, and there is also nothing to say against it. However, you will soon find that some kinds of content insertion are not possible in this view.

The Tags On View

This is our preferred view.

The **Tags On View** is also a kind of WYSIWYG but, by visualizing the tags, it also allows the insertion of content between certain xml-tags. Tags On View uses the same layouting as the **Normal** view, and also offers the same ui-controls, which have been described previously.

The Text View

In the **Text View**, you see plain text including all xml-tags with their attributes, displayed without any structure, which makes it inconvenient to use. You should avoid this view unless you know exactly what you are doing. There is a great danger of introducing editing errors leading to invalid xml, which can no longer be checked into the database.

Important settings

There are two important settings for the editor, which you have to set when you start to use it:

1 In the tools menu, select the Editor's options

2 In the general panel, uncheck the **Restore last open documents** checkbox

3 In the views panel, set the option to your preferred view (**Tags On View** is recommended)

Working with the Editor

When working with this editor, you are dealing with three main differences compared to 'normal' text editors:

1 The functional meaning of the content, which is reflected in the proper choice of xml-tags instead of dealing with layout.

2 The insertion point between tags, and its relation to the xml-structure

3 The semi-wysiwyg view, which is not exactly that what you see in the final document.

The functional type of content

Before typing in the content as you might be used in a normal text editor, you first have to get an idea of the functional meaning of the content that you are going to type in. Is it a procedure? Is it reference information? Is it a hazard? Is it a specific user interface element? etc.

When you know it, you insert the appropriate xml-tag from the **Insert Element** viewlet and then you fill in the content. For example, you have opened a newly created topic. You want to enter a heading and some reference information. Therefore, you insert the heading tag and write the heading, then insert the reference tag after the heading and write the content. You can also copy the topic heading into the reference heading or use a ui-control to do so (see “Heading” on page 35).

Text fragments can also be tagged afterwards by highlighting them and applying the appropriate tag. Currently, there is no support for turning complex functional tags into another e.g. a procedure into a reference, unless you insert the other functional tag, copy and paste the content from one

structure into the other while considering the different xml-tags being allowed in one tag but not in the other, and finally deleting the empty old xml-tag(s). We are working on giving you this automatically in the future, at least for the most common xml structures.

Insertion point and xml structure

Once you know the content and its functional meaning you still cannot enter content wherever you want. You have to follow rules that direct you to enter certain xml-tags only at certain insertion points. These rules are defined in the xml-scheme and XMetaL follows them strictly.

Let's take the procedure example, which you already know a bit. After entering the procedure tag, you see that it contains a goal tag at the beginning, followed by an optional explanation and the procedural steps. For example, if you want to enter a hazard, you have to position the cursor between the 'end 'Explanation' tag' and the 'beginning 'StepSequence' tag'. Here, you can insert the "SafetyInformation' tag' in which you can finally insert the 'Hazard' tag. Another position for a hazard is between the 'beginning 'Step' and beginning 'Action' tag. That's it! There is no other place for hazards. You might say that cannot be. How about when I need to insert a hazard somewhere else, or in a reference? Well, the answer is, you just can't. Related to this example, a hazard serves as a warning to the users before they are doing an action in a procedure. A reference serves to describe something. There is no need to warn anyone. The hazard that you wanted to enter here might be just a hint or note. Think like this when you miss a tag in a certain context. However, if you still think a tag is missing in a certain structure, we are open for discussion and will change it if you have the better arguments.

How to find out where to enter which xml-tags?

This is really not so easy unless you have the xml-scheme that controls this behavior in your head. In time, the most common xml-structures will become burned into your memory, even when you are just an occasional writer, because most of them are just logical. To fill the gap until this time has arrived, you can find the structure in the context help of the editor. The context help is described in the next section (see “Context Sensitive Help” on page 50). You also can position the cursor between tags and see in the 'insert element' viewlet, what can be entered here. This viewlet must be in the 'insert' mode to display the available xml-elements. The 'change' mode allows you to change

one tag to another, which is helpful for changing tags in paragraphs (e.g. 'Emphasis' to 'GUILabel' etc.). It does not work for complex tag structures as described above.

Aso, when inserting new tags, you will see that the major structural parts of this tag are inserted in addition with some instructions telling you what to do with these specific tags. This will help you to enter the content properly and not to forget anything.

Creating Variants with Variables

In the section "Variants" on page 36, we have seen variants at the level of information modules. They are basically copies but still have a relation to their originals.

Using variables, you can create variants of information modules without making copies of them. The creation of a variant takes place within the information module itself at its structural level: a list item, a paragraph, a note etc. Here, we can distinguish between variants, using variables for conditional texting or for variable text.

Variable text is the simplest form: you use a variable as a text replacement. The variable is defined in a topic or in the information product itself (see "The Property Viewlet" on page 38) and it is used in the reusable information modules.

Let's have a look to an example reference module, which consists of the following paragraph:

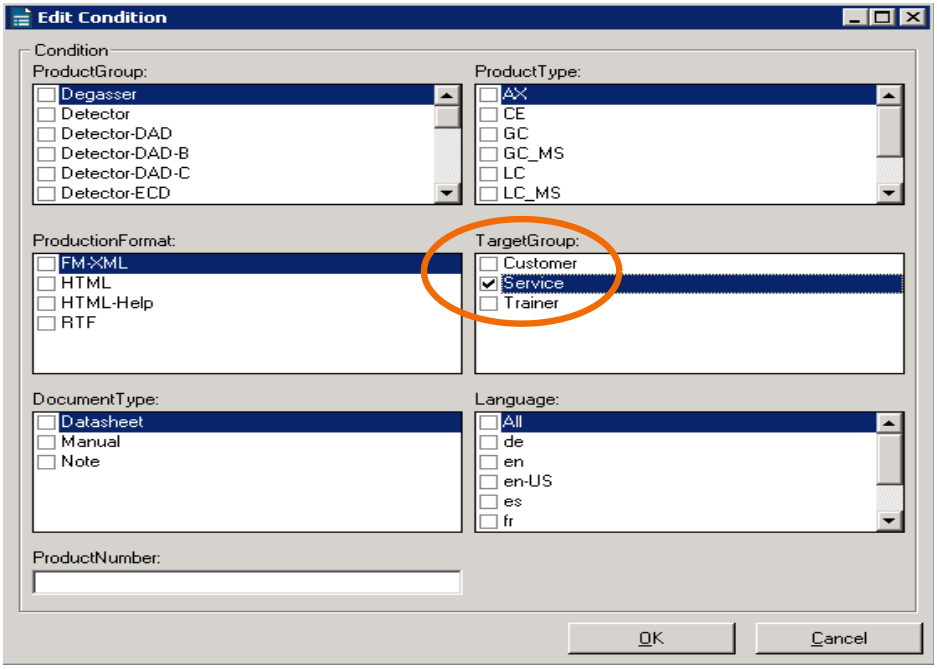
```
<Para>The maximum pressure of the pump is <variable
name="pressure"/> bar.<Para>
```

You might reuse this reference module in two different topics. In one topic, you define the variable 'pressure' to be 400 and in the other 800. This way, you can have two topics with the same reusable information module being embedded but in the final document showing up with two different contents: in one with 400 bar, in the other with 800 bar.

Conditional texting is a bit more complex. The principle is the same as for variable text: you can create different topics that all resuse the same information module but with different content in the final document. Hereby not only a word or text fragment is different, but the content of a complete structure can be different The condition that determines whether the

structure becomes part of the final document or not consists also of variables. It will be tested whether a variable has a certain setting. If the test evaluates to true, the structure becomes visible in the final document, otherwise not.

We will demonstrate this also with an example: assume you are writing a user manual and a service manual, in which you want to reuse a certain information module that fits to both manuals. Just one paragraph should not appear in the user manual. For this case, you assign a condition to this particular paragraph that is very simple to do for you as an author. You just activate the condition dialog for this paragraph, using a toolbar button, and check the **Service** check mark in the **TargetGroup**.:



In the information product that becomes the servcie manual, you define the variable **TargetGroup** as **Service**. Now, when you produce both manauls, only the service manual will show this paragraph. For the user manual, the condition will evaluate to false, which means that the paragraph will not become part of this content.

When you look at the condition dialog, you see that you can build conditions out of quite a few variables using different values. When multiple settings of variables are used, they will be logically connected with 'AND', and multiple values will be connected with 'OR'.

NOTE Avoid conditions that are too complex, to keep the overview of the expected result.

For those who are interested in the technical aspect of the implementation of the conditional texting, we will give here a short insight: the condition becomes an attribute to the structual element using an XPath-expression. The upper paragraph example will be read as:

```
<Para Condition="$TargetGroup=''Service''">...</Para>
```

This way, the condition can be directly parsed in a test statement of the XSLT-stylesheet that is used to generate the final document. You will see the condition as such in the preview viewlet and in XMetaL as tooltip, when moving the mouse over such a structual element.

The Graphics Editor

Authors who have been using Word or FrameMaker are familiar with the possibility of inserting images in their documents and annotating them. In FrameMaker, this can be done with callouts, which have the advantage that this text can be translated without being dependent on the underlying image.

In the new content management environment, and using a totally different editor, we didn't want to give up this feature, and have been looking for solutions. Since the textual content is based on xml, a solution was soon found using the svg-(scaleable vector graphic) format, which also is xml-based. One of the svg features is to xlink to an external binary graphic format. By doing so, we overlay the graphic with text, lines or shapes using the appropriate svg tags and actually get the same optical result as with the callouts of FrameMaker. Also, this text can be easily translated into any language, because of the Unicode capability of the xml-based svg format.

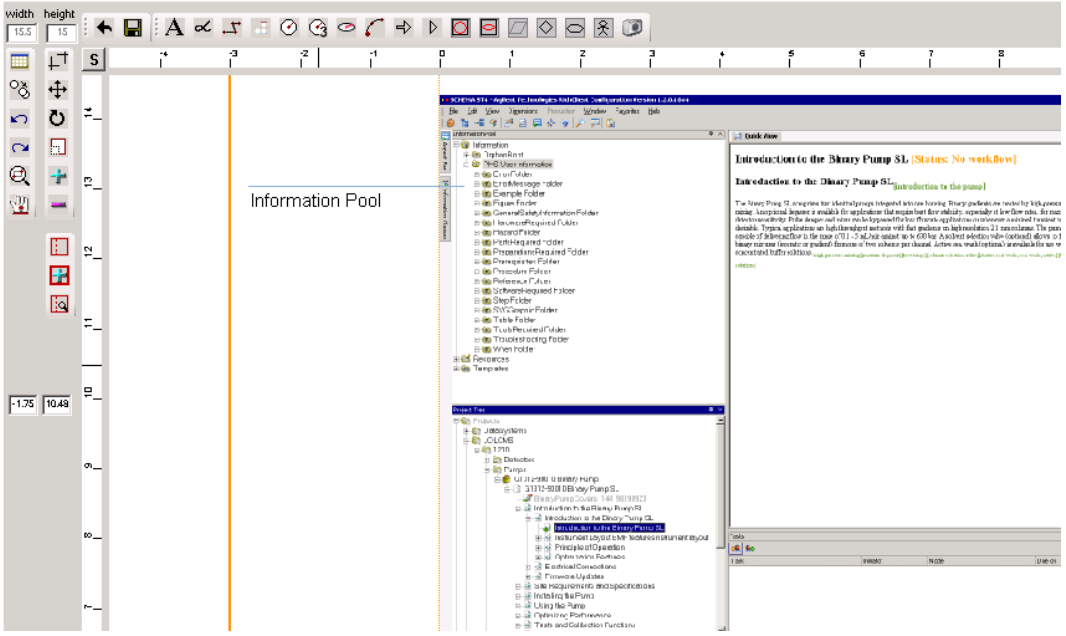
The next important question was how to take an image and annotate it. An appropriate graphics editor with the ability to save the graphic and text in a format such that the text can still be easily translated has to be found. At this

stage of the project, and even while I am writing these lines, an appropriate graphics editor is not available on the market. There have been some graphics programs on the market with the ability to save graphics in svg format, but the resulting output was overloaded with special svg tags and mostly not translatable. Also, these graphics programs were overloaded with features, designed for graphics experts, which makes these programs not really suitable for you as an author.

We decided to use an ActiveX component (AVAX from CivilTech) with a ready-made toolset of functions. We designed and implemented our own user interface around it using only the functions that we thought were useful for you. This has the advantages that this svg-editor does exactly what we want, and it can even be embedded into our xml-editor as ActiveX component, which makes this graphic editor very neat:

Para The svg-editor becomes active and you can annotate the graphic with lines and text for example: /Para

Figure



The svg-editor displays a frame, which exactly reflects the page of an Agilent manual including its margin. This allows you to layout the graphic with the callouts according to the manual page e.g. the image in the text area and the callouts in the margin area. It's nearly the same as in FrameMaker. Having this

frame and a ruler, you also get an idea of the graphic's size in mm or inch. This is quite important too, when the svg is also or only being displayed within an html page.

Last but not least, you also can use the SVG-editor for creating and maintaining e.g. formulas, flow diagrams and even use-cases for project managment (see also “Use Cases” on page 68).

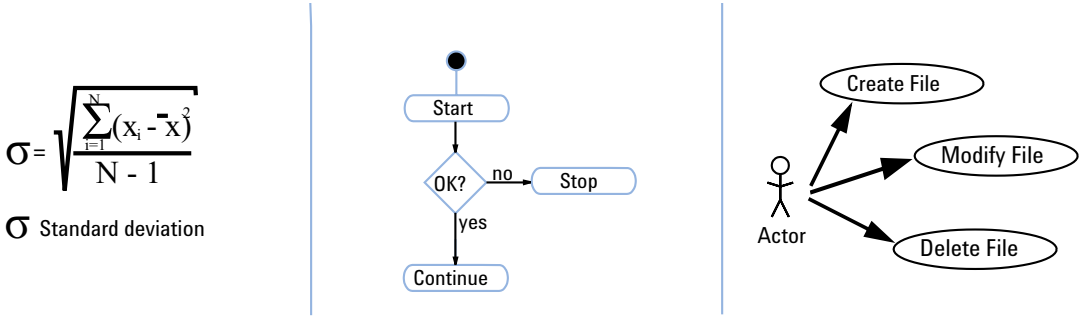


Figure 13 Other examples in SVG

Context Sensitive Help

The **Resource Manager** viewlet shows the context sensitive help. The content that is shown in this help is primarily dependent on the position of the cursor in an xml-tag. Information is given about this specific tag, usually also about the tag that has this tag embedded, and always gives general information about writing paragraphs. For example,when you have the cursor positioned in a button tag of a paragraph within a list, the help shows information about the button, the list and the paragraph. This information reflects mainly the rules and guidelines from the Agilent Style Guide. This means that whatever you have written can immediately be checked that it follows the Agilent Style Guide. Now there is no excuse that you didn't know it! It will permanently accompany you. Even when you close this viewlet, it will be reopened again when you move to text content. This is not to bother you, but to make you aware of how serious it is for us that you follow these rules and advice.

Additionally, you will find general help here about how to perform certain tasks in the editor, such as how to create a new topic structure or how to insert a graphic and annotate it etc. If you are missing certain help, or something is still not clear, then please contact the PHS user information team and let them know. Such problems can be corrected quite quickly.

In the context sensitive help, you will sometimes find a section called *What to do next?* This section not only gives you an idea of what you can do next while the cursor is at this specific tag position, it also performs it right away. For example, when you have the cursor in a paragraph of a list item, pressing the return key on the keyboard just inserts a new paragraph; *What to do next?* proposes to insert a new list item right after this one, or to insert a sublist, or to continue writing a new paragraph after the list. When you click on such a proposal, you are not directed to the **How to** section of the help, where this task is described, but the list item or the paragraph is inserted right away at the proposed position,and you can continue writing there. Isn't this neat?

We are open for proposals from your side to extend this feature according to your further needs.

Reuse and Ownership Concept

It might happen that you have opened a more complex structure for editing and you figure out that you cannot edit some text sections. The cursor has also changed to indicate that editing here is not possible. In this case, you probably have reused content that you do not own. What if you need to change this section? In principle, this is no problem. But because you are in a multi-user and multi-usage environment, you cannot just do it right away.

You have the following three possibilities:

1. Ask for the ownership of this particular information module or the whole content structure
2. Initiate a change request that asks the owner to modify the information module according to your needs
3. Eliminate the reuse aspect and create a copy of the text

We don't want to recommend the last option, since we are forcing the reuse and trying to minimize copies. The other two possibilities are suggested, and the appropriate workflows can be initiated outside of the editor (close the document first) on the ST4 user interface. Position the mouse cursor on the

appropriate information module or above in the structure hierarchy, and use the context menu (right mouse click) to start the appropriate workflow. In the first case (ask for ownership), you can continue editing after accepting the ownership change. In the second case (change request), you should double-check the result after receiving the message that the changes have been done.

HINT To get the changes done the way you want, ask for the ownership and do it yourself.

There are two reasons for changing a reused information module. Either it needs to be corrected or improved, or it needs to be extended to fulfill more or other needs. In the first case, the modifications can just be done. In the second case, a variant has to be created to fulfill all reuse needs, the old and the new ones. Variants can be created by either using a variable or by applying conditional texting. For both cases, the appropriate variables have been set on the topic or information product level (see “The Property Viewlet” on page 38). If the variable has been already set, its content will now be shown, otherwise not.

Be aware that this information module can be or mostly has been reused somewhere else. The usage info viewlet of the database user interface will tell you where. However, don't be afraid when you have changed something. It does not immediately influence the reuse in the other context. The reuse is bound to a fixed released version of this information module and when you modify it, you will create a new working version of it when you check it into the database. All authors who have reused this - now modified - information module will be informed that their version is no longer the current one. They will see suspect links in their task viewlet (see “The Task Viewlet” on page 39) to indicate that they are reusing information modules for which a newer version now exists. They can follow these links to the reused information modules and update to the newest version, if it's appropriate.

The reuse and ownership concept have some impacts to the processes. This is not really visible to you, but you should be aware of them to better understand why some processes and handling are as they are.

Imagine you have reused a couple of information modules, some even more than once. Now you have checked out for editing a substructure of your information product, which contains all these reused information modules. You are not able to modify those that you do not own. We've seen this before.

Those that you do own, you can edit. Assume that you own some information modules, which you have now reused multiple times, for example, a certain step in a big procedure, at the second, sixth and tenth positions. Now, you are modifying this particular step at position six and ten because you think it should be described differently. From your point of view, there is no problem, but from the cms's point of view, it is a challenge, because in the database this step exists only once. Remember, it has been reused three times. The check-in process has to make sure that new entries are generated for those two steps that have now been modified by you.

Another scenario would be that you want to copy a reused information module and modify it for your purpose. This is a typical process in a text editing program, but also a challenge in this environment. When you copy a reused information module you would also duplicate its ID, which must be unique in the database for its identification. The xml would become invalid. The cms process has to consider this and handle the copied information module like a newly written one, which means it will receive its own ID when it is checked into the database. In such cases, please check whether this can be better solved with a variant.

These few examples demonstarte that operations that are simple in a text editor may become complex in a multi-user, multi-usage environment.

Some General Rules and Tips

When writing, you do not have to think about yourself, but about the others who will have to read it. Do they understand what you are writing? Keep the phrases short. Each word that you write will cost about 25 cents to translate. Also keep in mind that your phrase might be reused in a different context and in a different information product.

If you want to put a phrase or a word into a special format, then ask yourself why you want to do this, and instead use the appropriate tag that represents the function that you had in mind for this word or phrase. For example,if you want to highlight a word that represents a label of a field in a graphical user interface (GUI), then tag it with the GUILabel tag and not just with the emphasis tag. In this case especially, it will have an impact on the translation, because terms that are tagged with GUILabel will show up in the printed documentation in the appendix in the form of a translation dictionary for

those languages where the graphical user interface of the software has not been translated. Terms that you don't want to have translated at all, you can tag with the **ReservedWord** tag.

These example visualize how important it is always to use the correct functional tag, because it is not only the final layout that can be affected. The context-sensitive help in the XMetaL editorcontains detailed description about the specific functional tags within a paragraph.

Here are some more hints that help to make your work more efficient, and optimize your quality:

- Use the special 'insert graphic' button in the toolbar for inserting new graphics.

  Don't use the resources viewlet of teh XMetaL editor to do so. The graphic must have been imported to the database before you can insert it with this button. Actually, you only create a link to the graphic in the database, even when it is displayed to you. That's why it does not work when you load it from the file system using the resource viewlet.
- Use the spelling checker (tools menu) and the validation tool (toolbar button) before checking in (save toolbar button or ctrl s) your editing.
- Check in (save) your work from time to time, also during editing
- Close only the document, but not the editor, when you have finished editing

  Reopening another information module for editing will be faster the next time.

Considering this, and the guidelines given in Agilent's writing style guide, you will become a perfect user-information writer. However, the Agilent Style Guide is not totally comprehensive, and language skills are also quite important!!

6
My First Book

The creation process of an information product is described.

Now we are at a point where you have all the knowledge to create your first book, as I have done it with this one. Mine was more a book, yours will mostly be a manual or an online help.

Creating the Book

You start with creating a new book, which can be completely new, like this one, or derived from an already existing information product. In the first case, you would take a template to start with and copy it into the project. Appropriate templates are available in the **Templates** folder of the information pool. In the second case, you would create a copy or variant from a similar information product, which you can then modify according to your new needs. The variant is the better choice when the new information product is either very similar to the existing one or the new one can be seen more or less as a subset of the existing one. Also when creating a copy, links to topics will become broken. For example, our Agilent 1200 Series user manuals are a subset of our related service manuals, and therefore they can be seen as a variant of them (see "Variants" on page 36).

Now you can delete unwanted topics or add new ones to the current structure. You can embed reusable information modules into the topics simply by dragging and dropping them from the information pool to the topic in the structure. If a topic resides at the wrong position in the structure, you can just move it to another position, including its substructure.

Once the structure has been set up more or less complete, you can start to fill it with content, either by further reusing existing content whenever possible, or by writing new content using the editor. Within the editor, you can still delete, add or modify topics and embedded information modules. Try to take advantage of all you've learnt from the previous chapters.

Reviewing the Book

Once you've entered all the content, you'll want to get it reviewed by someone else. Either you want to have it checked for technical correctness or for good English, as in my case, since I'm not a native English writer.

You have to start the review workflow by invoking it with a right mouse click on the information product in the ST4 user interface, and selecting the appropriate menu item in the context menu. The reviewer will receive the task and can start the review.

Usually, the review is done offline and the reviewer should also receive a PDF of your information product. The reviewer will enter the corrections and comments in the PDF, which the author then has to use as basis for applying the appropriate modifications in the information product in the database.

In my case, the reviewer also had access to the database, and could do the modifications directly. In the editor, he turned on the 'track changes' tool, which allowed me later to see what he had modified. He also entered his comments either directly in the editor using the comment tag or by entering the text in the comment property field (property viewlet) of the information module. The latter is useful to make comments related directly to the information module.

As a last step, after entering or accepting the corrections, you have to terminate the review workflow, which brings the information product into the 'ReadyToRelease' state and allows you to start with the publication process.

Publishing the Book

Now you can do the publication process, which starts with the production of the book. If it is an online production (html, html-help), you are usually finished; you just have to copy the resulting files to the final media.

If it should become a PDF, you do an xml production for FrameMaker.

After the ST4 production several more steps are necessary to get the final pdf:

1. The xml file from the production has to be transformed into an xml, which is appropriate for FrameMaker: a few xml tags have to be converted and the table of content and the index has to be prepared.

   You will do it by invoking a publishing preprocess routine on this information product. Here, for example, you can also define how many levels the table of contents should have
2. This converted xml-file has to be imported into FrameMaker using the appropriate language template.
3. The FrameMaker content has to be checked and slight adaptations have to be made: correction of picture sizes and page breaks for example.
4. The PDF is creared using the appropriate settings for printing or electronic publication.
5. The PDF is printed if required
6. The PDF is finally released in SAP.

Localizing the Book

If you want to have your book localized, you can give the content of your information product to a localization service. The cms supports you by either exporting the complete content or by exporting just the changes since the last localization. The latter is very useful, because you might have reused information modules for which a localization already exists. Your localizer will not translate this content again, but has to identify those parts for which you also have to pay. You can save this money by not sending content that has already been localized and not changed since.

The database export of the cms is optimized for import directly into the translation memory system. No further settings must be done on this side; the localization process can start right away.

Once the localization is done, you will receive the localized files back from your localizer. You can import them directly into your database automatically aligned with the same version of the English information product.

Now you are already in the position to start the publication process for your localized information product. However, it is advisable to start with a test production, and see where slight modifications have to be done in the database content. Especially because of different text length, some svg images may need to be newly aligned. You are also allowed to open the localized content in the editor for modifications on this particular version.

NOTE In this case, *NO* new version is created when the modifications are checked in!

7 View into the Future

We show what else can be doen with such a cms without much effort, by reusing its principal potential.

We've brought the cms to a state where content can be entered or migrated into it, checked out and in for modifications and translation, and various information products such as PDF via FrameMaker, html-help and various html formats can be generated. The system supports you with friendly user interfaces, an integrated style guide, and workflows.

However, there is always potential to improve the system and extend it for other related purposes.

In the following, we will give you some idea of what we have in mind for this system, to further improve its usability, and to show the potential of such a system in areas other than those for which it has been designed.

Controlled Authoring

Controlled authoring means writing in a controlled natural language, which in our case is English, and to reuse as much content as possible. Controlled English would be a subset of standard English with a restricted syntax and restricted semantics described by a small set of construction and interpretation rules. The major goal of controllled authoring is to make it simple and understandable to non-English speakers, and to minimize localization costs.

We will not be able to modify all our existing content to controlled English but we can implement mechanisms to motivate our authors to use controlled English, and to reuse as much as possible in their future writing. In this respect, the focus is set to the reuse aspect, and the use of standard terminology.

Optimize Reuse

Authors have to be motivated to reuse content as much as possible. This also means that we have to provide mechanisms that make it easy to find content that has the potential to be reused in the particular situation. The current impementation does not support this adequately, because the author has to search for reusable content. It would be better if the systems were to look for it and propose such content. Once an author enters text in the editor, the system checks the database for similar text and proposes it for reuse. This has the advantage that text can be reused for which alread localizations exists. If the proposed text is appropriate, and does not change or weaken the meaning, it can easily be used instead of what the author was going to write. The system even goes a step further: it also checks the database for information modules with this text and proposes it for reuse. The author does not have to actively search for it.

Terminology Support

PHS already has started to set up a terminology database that is common to PHS and is already used for localization. It guarantees that a certain English term is always translated to all languages in the same way, and must not be translated again and again. However, it is highly desirable that all authors are also using the same English term in this sense. This would not only further improve the quality of our user information, it would also further decrease the localization costs.

This can be done by allowing the author access to the terminology database to look up the terms. Even better support can be given when the terminology database is connected to our cms to support our editor. Assuming you have written your content in the editor, you would then just do a terminology check in addition before you do the spelling check. The terminology checker would search for terms in your content and compare them with those in the terminology database. Whenever it finds a term that should be replaced by the proper term from the terminology database, it will propose it for you and you can simply replace it.

Special documents

In principle, any document can be created, managed and produced by this content management system. There are some type of document for which we can see an almost imediate benefit when they are also part of the system.

Service Notes

Service notes very often contain content that could be reused in another context. Service notes have been entered into a warehouse from which they can be downloaded by the service engineers worldwide. In addition, a service note viewer is created out of all existing service notes as a very convenient tool for the service engineers. This currently involves some manual work.

By using the cms for entering service notes

- you can take advantage of the usability aspects that the system already offers,
- you can automatically create the service note viewer,
- you can upload the service note to the warehouse.
- The content of the service note normaly becomes part of the service manual in a later version, which makes the content of the service note reusable to a high degree.

Application Notes

Application notes are currently written by authors in Word, then imported into Quark Express (a publishing tool) and finalized for publication. There is also a desire to reuse parts of this content.

Why not, then, also enter application notes directly into the cms?

- you can take advantage of the usability aspects that the system already offers
- you can reuse parts of the content of the application notes

- you can export the application note in an xml-format that is appropriate for import into Quark Express or InDesign for final publication

Technical Notes

Technical notes are similar to service notes from the content point of view. However service notes are internal documents being delivered and read only by the servcie organization. Technical notes will be given to customers which implies higher quality to the writing style and layout. From the cms point of view its just a similar information product produced with a different layout.

Slide Shows

Slide shows will be created to a high degree with PowerPoint. Providing the content of these slides from within the cms allows the user to much better organize and reuse the content for all the various slide show presentations. A production specialized to create PowerPoint format will then generate the slide shows.

Project Documentation Management

Our cms has been set up in such a way that we can also write and manage our cms project documentation in the system. We already have done this for the requirements and use cases. As we're now in the state to do it for all documentation, we will do the complete project documentation from now on.

System Requirement Specification (SRS)

A requirement is defined as a condition or capability to which the system must conform.

Once a requirement is formulated, it will be constantly discussed and modified until an agreement has been found on it, that it can be approved for implementation.

In our cms we've set up the requirement model such that

- you can take advantage of the usability aspects that the system already offers
- the requirement can be formulated within an individual xml-structure
- discussion points can be entered
- change tracking of the requirement modifications can be set up and followed up on
- A set of properties (Category, Approval Status, Priority, Implementation Status, Effort, Risk, Target Release, Source, Specifier) controls the current state of the requirement

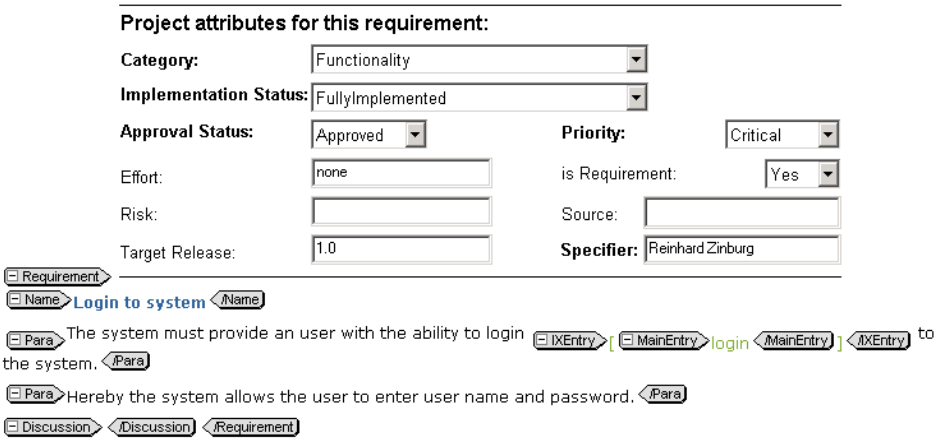


Figure 14 Example: Requirement

Requirements management is a systematic approach to finding, documenting, organizing, and tracking a system's changing requirements.

A certain requirement management can already be done with our cms by giving access to the cms to all participants who are involved in the requirement management process.

However, this is a cost issue. Each access must be licensed. An ideal solution here would be to use a so-called Web Client, which allows access to the cms to multiple users via intranet, and includes licenses for up to 50 unnamed users. It is not as cost-intensive as individual licenses. The Web Client has to be adapted to our requirement-xml-structure.

Use Cases

Use cases are used to organize the functional requirements. Instead of creating a bulleted list of requirements, use cases organize them in a way that tells a story of how someone may use the system. This provides for greater completeness and consistency, and also provides a better understanding of the importance of a requirement from a user's perspective.

In our cms we've set up the use case model such, that

- you can take advantage of the usability aspects that the system already offers

- the use case can be formulated using a use-case-specific xml structure like a form
- the use case diagram can be drawn according to the UML standard using the svg-editor
- a set of properties (Priority, Risk, Approval Status, Implementation Status) controls the current state of the use case

This way use cases can already be entered into our cms and documented e.g. as PDF.

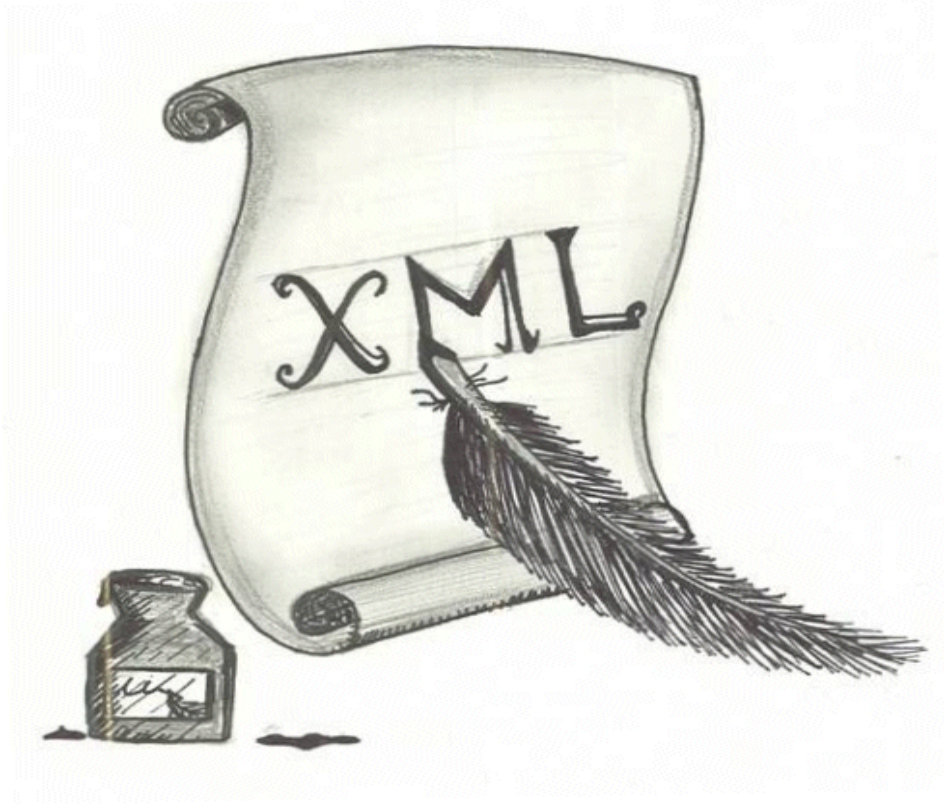


Figure 15 Example: Usecase

External and Internal Specification Requirement (ERS and IRS)

The ERS and IRS describe the product in more detail. Agilent uses a special template (e.g. Word template) for documenting this specification. We have adapted this document template into our xml model to allow these specifications to be written into the cms based on the same structural ideas.

Thus

- you can take advantage of the usability aspects that the system already offers
- you can reuse parts of the ERS content in the user information documents

  In this respect, you, as an author of the user information document, either create a variant of the ERS part for reuse, or improve the content of the ERS itself to reuse it directly. This also would improve the quality of the ERS itself.
- the information flow is improved, because ERS writers and authors are informed by workflows when something changes related to the content of the information module being commonly used in the ERS as well as in the user information.

  This is very useful, especially during software development, to have the ERS mostly updated.
- version control is already part of the cms

Test Cases

Test cases describe the actions that have to be performed to test the functionality of a system.

They should also be written according to a specific template. We have also considered this template in our xml model, especially by reusing the prerequisites and procedure, which are also a major part of the test case.

The PHS-QA-department already uses test cases for software tests based on an xml template.

By incorporating this into our cms

- you can take advantage of the usability aspects that the system already offers

- you can reuse parts of the test cases (prerequisites, steps etc.) in other test cases and in the user information (e.g. procedures)
- version control is already part of the cms
- testcases can be used as templates, of which the tester makes copies to use as the testcase for entering the test results
- a WebClient, as proposed for the requirements, would also be very suitable for this purpose
- test results can be presented in an intranet using a specific html production.

  An html intranet for this purpose has already been developed and is in use by QA. It can be used as a basis for this special html production development
- test reports of any kind based on the above html production can be created
- usage of electronic signature can be included

  A special module for the database (audit module) is available to allow this feature

Putting it together

Project management is not complete if all its individual project documents remain individual. They have to be linked together such that one can trace from the use case and requirement via the ERSs to the testcases and results and back from a testcase to the use case or requirement. This is possible by setting up an html production that creates an intranet that showing all project documents in html- or PDF-format and linking them together as described above.

This linking will be done automatically by putting the project information products into appropriate groups, with certain properties and rules. The authors just have to create the appropriate information product (use case, requirement, ers, test case etc.) in the right group and the linking will be done automatically when the intranet is updated. This intranet can be enriched with project status information and even a Gantt chart, which would make this intranet the perfect platform for communicating project information. A prototype, showing how such an intranet could look, is available and can be seen and explained by me (Reinhard Zinburg, contact).

Of course, by investing more, the intranet could become a portal in a future step that would allow the project documentation and information not only to be viewed, but also to be accessed and modifed directly from the intranet. Portals with ST4 cms and Microsoft's sharePoint server are currently in the trial stage.

Knowledge Database

By entering all available information into this database, it would become a very powerful knowledge database. Ontology techniques could be used to find information.

Also, other departments, such as the Customer's Training Center, can find and reuse information for their needs more effectively.

Index

U

V

W

X

www.agilent.com

Motto

The aim of education is not knowledge but action. Herbert Spencer (1820-1903), engl. philosopher & social science.

Printed in Germany
2nd Edition March 2010

1234-56789

Agilent Technologies