EPK-fix: Methods and Tools for Engineering Electronic Product Catalogues*

A. Knapp, N. Koch, M. Wirsing (LMU München)
J. Duckeck, R. Lutze (mediatec GmbH, Nürnberg)
H. Fritzsche, D. Timm (TU Dresden)
P. Closhen, M. Frisch, H.-J. Hoffmann (TH Darmstadt)
B. Gaede, J. Schneeberger, H. Stoyan, A. Turk (FORWISS Erlangen)

Abstract An electronic product catalogue (EPC) is a computer controlled information system with multimedia product presentations and navigation facilities. The paper presents the results of the EPK-fix project for the systematic construction of EPCs. These include a software engineering process model, a high level specification language for EPCs, and an integrated set of tools supporting the entire EPC development process. The EPK-fix process model supports the classical development phases: requirements analysis, specification and design, implementation, and test. In each of the phases emphasis is put on the particular multimedia aspects, human machine interaction, production of prototypes, and the quality of the produced documents.

Introduction

Electronic Product Catalogues (EPCs)[1] are computer controlled information systems with important multimedia (especially visual) product presentations and navigation facilities. They are almost always equipped with a shopping bag administration feature.

EPCs are an inexpensive alternative to paper catalogues, but a high quality design is still related to elevated costs, because there are no appropriate production tools available. There are *catalogue providers* introducing a multimedia presentation to the market, *catalogue developers* designing and producing EPCs with the assistance of software and multimedia experts, *testers*, and *users* or *end-users*.

State of the art technologies to produce EPCs are still far from being easy to use and efficient and they show serious weaknesses. New methodologies and specific tools for EPCs are needed. They must support the complete *life cycle* of EPCs starting with the analysis of the catalogue providers requirements, continuing with the catalogue design up to the functional tests. These tools have

* This work was supported by the BMBF project EPK-fix (Förderkennzeichen 01 IS 250). Corresponding author: N. Koch, Institut für Informatik, Ludwig–Maximilians–Universität München, Oettingenstraße 67, D–80538 München, `kochn@informatik.uni-muenchen.de`

[1] German: EPKe (Elektronische Produktkataloge)

to be easy to use, reduce the amount of EPC development time, and permit a low-cost production of catalogues. These are neccessary prerequisites for the acceptance of a new system, especially in small and medium size organizations.

We present the results of the EPK-fix project for the systematic construction of EPCs. Methods have been developed and a collection of integrated tools (RASSI, SASSI, GASSI, TASSI) has been designed for efficient specification, production, and validation of EPCs. The requirement analysis is carried out with the help of structured interviews which are guided and recorded using the first tool. The result of this analysis is the base for an EPC design written in the specification language EpkMl. EpkMl is a high level HTML-like language that is particularly designed to support the description of EPCs. The second tool supports the specification construction with a set of specialized editors. The third tool automatically generates different catalogue versions in Java from the EpkMl specification. Finally, the test assistant helps to ensure the quality of the produced catalogues. It provides static and dynamic test strategies as well as strong support for manual testing of media objects.

The first section of this paper outlines the state of the art of EPCs, the development process, and the resulting architecture of the EPK-fix system. Sections two to six describe the specification language and the tools respectively. In the last section some conclusions are delineated.

1 Developing EPCs with EPK-fix

The analysis of about 40 EPCs [6] has demonstrated that they go far beyond paper catalogues with cross references. They offer services like search features, demos to show how to use the catalogue, games, query language, enquiries through telephone communication and fax, or on-line ordering. The constituents we identified in each EPC are: *structure*, *layout*, *direction*, *database*, and *services*.

- The *structure* is the skeleton of the catalogue; it comprises a graph or hierarchy of themes and pages.
- The *layout* is the static description of pages and their contents.
- The *direction* describes the dynamic facilities for pages and themes that allow for user interaction and the navigation through the catalogue.
- The *database* component provides all the information about offers, in such a way that it can easily be searched, exchanged, and maintained.
- The *services* add comfort to the EPC allowing i.e. administration of orders, user registration, access to help functions, online communications.

We observed that working with these EPCs can mainly be divided into installation, presentation, search, selection, and order steps. Depending on their relative importance , we distinguish between the following catalogue types: *Presentation*, *Search*, and *Order* catalogues; see [6].

The Development Model. The development process for EPCs requires an informal preanalysis followed by the phases: *requirements analysis, design, implementation, and test.* For each theme a tool has been developed considering the above mentioned aspects *structure*, *layout*, *direction*, *database*, and *services*. Using a *formal specification* based on a specification language it is possible to take into account the application requirements and to avoid inconsistencies among the different tools. Characteristics of the EPK-fix development process are the emphasis put on the multimedia aspects, the human-machine interaction, the production of prototypes, and the quality of the produced documents. The resulting process is a kind of spiral model [2] that leads to the final EPC through successive revisions and refinements.

Requirements Analysis. Human dialogue is central to the analysis process in engineering. The fundamental analysis activity is to carry out *structured interviews*. This interviewing process is divided into three major steps [8]: preparation, interview, and edition. During *preparation* a questionnaire is assembled considering all aspects (see Section 1) of an EPC. The selection of the questions is guided by predefined generic checklists. The *interview* is a complete (audio-)recording of the conversation between the EPC developer and the catalogue provider. Arbitrary multimedia information are attached to the corresponding questions and answers of the interview. Finally, the *edition* constructs the resulting analysis document from written (transcribed) notes and acoustic data.

Design. The informal catalogue description recorded in the analysis document provides the basis for the design of an EPC. The *catalogue developer* carries out his work with the help of appropiate *editors* which generate automatically a formal specification of the catalogue. Work starts with the analysis document specifying all the aspects and details of the intended catalogue. The developer supplies the structure and the layout of example EPC pages (or templates).

Integrated Software Generation. A general advantage of our approach is the speedup given to the otherwise time consuming and error-prone implementation phase. A generation assistant produces a prototypical EPC version with additional interfaces to communicate with the other development tools. These interfaces provide *capture* and *replay* facilities that can, for instance, be used to present a catalogue prototype to the developer in exactly that state which is refered to by a given analysis document. For testing purpose, catalogue elements can be addressed by their unique identifier (provided by the design tool). Event handling of the user interactions is done by a separate module allowing to treat simulated user input generated by the test assistant in exactly the same way as real end-user input.

Testing. Quality assurance is a central point of our development process. We aim at complete not only sample testing. In the generated EPCs the large number of media-objects are tested automatically via rules. Dynamic tests are used to ensure correctness of time-dependent multimedia processes. Manual validation of the layout is supported by test agents. The test reports are fed back to the other development steps.

The Architecture of the EPK-fix System. The EPK-fix system components rely on the formal description language EpkML. It integrates the following four tools: RASSI, SASSI, GASSI, and TASSI.

- EpkML is a specification language that makes the description of the static and dynamic aspects of EPCs possible.
- The Requirements analysis ASSIstant (RASSI) supports the informal recording of information (text, sound, images, video) that results at the requirements analysis stage based on structured interviews.
- The Specification ASSIstant (SASSI) is responsible for EPC design based on the results of the RASSI tool and generates an EpkML specification. The catalogue developer is assisted by efficient and powerful editors.
- The Generation ASSIstant (GASSI) translates the EpkML description specifying the EPC into a general programming language (e.g. Java).
- The Testing ASSIstant (TASSI) performs static tests on the catalogue description in EpkML and a dynamic validation on the EPC generated by GASSI, using test data especially prepared for that purpose.

2 The Specification Language

The design of the EpkML language was guided by basic aims like easiness of learning and extensibility as well as more EPC-specific considerations like the integration of database features or navigational support. EPCs on CD-ROM and on the World Wide Web (WWW) create an alternative channel to traditional paper catalogues for product sale and service offer. Since these catalogues may coexist it was our goal to develop a declarative language that allows an easy common specification.

EpkML is HTML-like. It is defined as an instance of SGML (ISO-standard 8879, [4]). The Standard Query Language (SQL) is integrated into the language for easy access to relational databases. EpkML has primitives for navigation flow and allows connection to external languages via applets like in HTML. The language integrates a conceptual view of EPCs, this means a structured, annotated, multimedia, and highly automatic front-end to a database (see Section 1). We restrict ourselves in this paper to only a few aspects of the syntax and pragmatics of the language EpkML that distinguish it from other mark-up languages. A detailed description including a formal structural operational semantics is presented in [5]. Appendix A shows one page of the tourist catalogue example and its EpkML specification. Below we describe some characteristics of the language and reference the lines of the specification.

Structure. Products are organized in hierarchies, so-called themes (`<theme>`, 01). The developer defines the products belonging to these themes (`<extension>`, 02) and the presentation of these products (`<template>`, 07). These templates are to be filled with actual product data obtained from the database. Whenever a special layout is desired, it is possible to define `<exceptions>` for those products with their own `<template>`. A hierarchical structure is achieved via the definition of sub-themes for a `<theme>`. It is also used to generate automatic navigation facilities supported by the command tags `<next>`, `<previous>`, `<up>`, `<down>`, and `<back>`, which branch to the next or previous theme in a given hierarchy, to the first one below or above, or back in the history of visited themes, respectively.

Services. EPKML includes for standard functionalities of a catalogue the following tags: `<table-of-contents>`, `<search-form>`, `<registration-form>`, `<question-form>`, `<shopping-bag>`, `<shopping-list>`, `<order>`. In the example a `<registration-form>` (44) is mentioned, that permits the personalization of the catalogue. A `<shopping-bag>` is a template for a list of products that have been selected as "products to buy". For more details see [5].
Audio-Visual Elements. The visual (layout) features of EPKML are a superset of those of HTML. Introducing e.g. `<window>` (08) we make EPKML window oriented instead of screen oriented. It is worthwhile to mention `<flowbox>` that distributes layout elements surrounding e.g. an image or text. Multimedia is integrated by adding the time-dependent elements `<video>` (15), `<slide-show>`, and `<audio>`. All these elements may be customized in advance using `<style>`s (08) by defining `<stylesheet>` for them.
Interaction. A large part of EPKML is concerned with interactive elements, like `<button>` (28, 41) or `<pulldown-menu>`. Interactive elements are provided with a specifiable method, e.g. `<on-click>` (30, 43) for `<button>` or `<on-option>` for `<browser>`. For the navigation through the theme structure, there are precustomized elements with standard behaviour like `<previous-button>` (38), and `<next-button>` (40). Of course, this behaviour can be changed or extended.
Database. Database access is achieved via the `<sql>` tag (03, 20). Text within the scope of this tag must be written in standard SQL.

3 The Requirements Analysis Assistant

The *Requirements Analysis Assistant* (RASSI) is a software tool that supports the task of interviewing. It contains five integrated submodules, that serve to manipulate and convert the basic object types *checklists*, *questionnaires*, *interviews*, *documents*, and *protocols*. Checklists enumerate all important aspects that have to be addressed during EPC design and development. Interviews with the catalogue provider are performed on the basis of quetionnaires resulting in protocols. The following are the RASSI submodules: The *checklist editor* manipulates new or predefined generic checklists. The *questionnaire editor* is used to assemble a questionnaire from a checklist by formulating questions and adding explanatory documents. The *interview assistant* performs a complete audio recording of the interview, allows for synchronously written notes, and links arbitrary multimedia material. The *protocol editor* supports revisiting and combining mutiple interviews. Finally, the *presentation assistant* generates an analysis document. RASSI is well-integrated into the EPK-fix toolbox: the chosen format of the analysis document (HTML) ensures immediate document exchange via the WWW enabling distributed workgroups to interact efficiently.

The *Tourist Catalogue Example* (Figure A) illustrates the development steps of EPCs throughout this paper. For information elicitation in the *system analysis* phase, a questionnaire has been created starting from a predefined checklist. As shown in Figure 1 the questionnaire contains the topic "Screen elements for city pages" which is part of the main topic "Layout and micro-direction". The

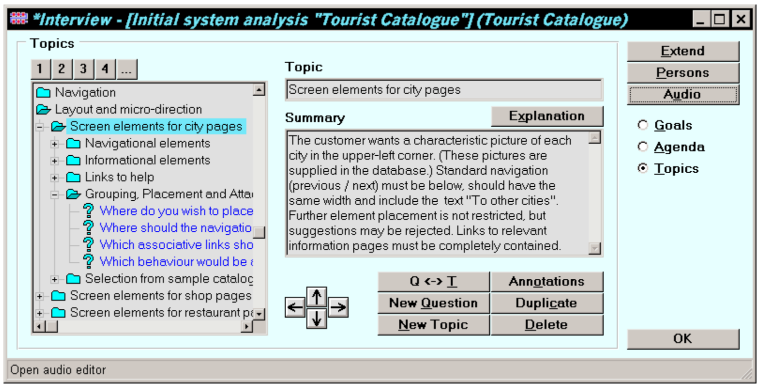

Figure 1. Main window of the interview assistant in RASSI

initial checklist mentioned the topic "product pages" which is now replaced by "city pages". The topic "Grouping, placement, and attachment" was discussed with the catalogue provider asking four specific questions. Explanations and multimedia annotations can be attached without any restriction. After the interview, the essential information from lower-level topics is summarized. The result of the final analysis document (HTML) can be viewed with any standard WWW browser.

4 The Specification Assistant

The Specification Assistant supports the transformation of an informal description of an EPC to a formal specification. A qualified developer uses SASSI's set of powerful editors to compose an EPC according to the information gained by the requirements analysis. SASSI is based on an intuitive graphical user interface, it is implemented in Smalltalk, and its architecture is based upon the design evolution presented in the DIADES user interface builder [3].

Main features of the SASSI component are to support a complete specification of the analysis information, to protocol which analysis objects led to which specification objects, to make sensible assumptions where analysis information is incomplete, to provide version information in order to make recovery possible, and to generate syntactically and static-semantically correct EpkML output. These features are supported by an object-oriented class library of specification object classes which reflects the EpkML elements. When specifying an EPC the user simply composes a hierarchy of specification objects similar to an abstract syntax tree which can then be used to generate EpkML output.

SASSI can handle all catalogue aspects (described in Section 1) with the following editors: *structure editor*, *layout editor*, also responsible for specifying direction and integrating services, and *database editor*, where database access can be formulated using SQL statements. The SASSI application starts with the structure editor window showing a view of the predefined initializing catalogue structure, which is subsequently refined according to the example. On double-clicking a structural node the layout editor opens a window displaying the representation part of the node. Layout elements can be placed according to the RASSI analysis of customer needs, making up the look of the catalogue pages. Entries, e.g., information describing the various products, images or even animations, will be taken mainly from the customer product database and have to be referenced properly using the SQL editor. The conception of the layout editor is similar to DTP layout applications. In combination with the structure and the SQL editor the EPK-fix approach goes beyond ordinary DTP applications.

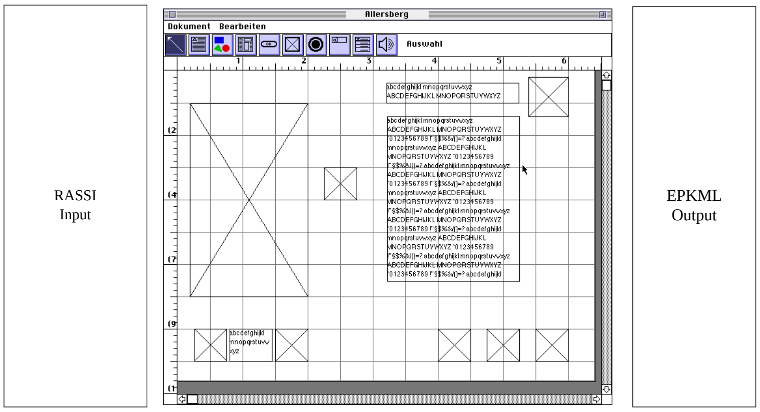

Figure 2. The SASSI user interface

To reference the requirements analysis input there exists an additional window in which analysis units coming from RASSI are displayed. Combined with a third window transcribing the generated EpkMl specification, this results in a three-window approach (see Figure 2) similar to [9] and the split-screen approach used in Oberon [7]. Working in either of the two *editor* windows, the user simply has to highlight the paragraph in the RASSI window corresponding to the work he actually performs, and all objects created will reference the identificator of this unique analysis information unit.

5 The Generation Assistant

The purpose of GASSI is to generate catalogues based on their formal specification. Targetted output platforms include a paper version (i.e. LaTeX), an HTML version and an EPC implemented in Java. Within the scope of this paper, we will concentrate on the presentation of the latter's generation.

The generation of EPCs relies on a library of extensible generic classes (that are reusable, and reliable due to their automated testing by TASSI) providing

EPC-specific components ranging from simple layout elements up to modules that perform *services* (e.g. online-help ordering-facilities). The generation of software implementing a formally specified multimedia system is done by analogy with compiler phases [1] as follows:

Parsing comprises lexical and syntactical analysis of a given specification. An EpkMl conform document defines a tree structure of EpkMl elements that can be parsed by a standard SGML parser yielding a more easily processable text output. GASSI calls the parser *nsgmls* and uses its output for further processing.

An *Intermediate Representation* of syntactically correct specifications allows for semantic analysis and optimization. The internal representation of a specification consists of *specification objects*, that are nodes of a tree reflecting the specifications structure. Each node contains an SGML element allowing to access the attributes of this element which had been set in the specification.

The objects are specific to the EpkMl element they represent by providing specialized methods to generate code implementing the desired layout and behaviour for this element. These methods encapsulate implementation details like different classes of the library an EpkMl element is mapped to depending on its content. Besides, methods for restructuring the tree are available.

Sourcecode Generation is done according to the syntax of the output format and the results of the specific optimization. The basis of the code generation is a class library realizing common features of EPCs which are used for a specific implementation either by direct instantiation or by subclassing and instantiation of the new subclass. *Specification objects* have access to a mapping table defining which library class has to be used for the elements implementation. For these classes miscellaneous Java-syntax-conform expressions can be retrieved or composed, e.g. unique Java identifiers; constructor calls; variable declarations or templates for subclasses.

The structure of the specification that implies a default structure for the EPC by the nesting of `<themes>` and therein contained instances for `<templates>` is used to build a module realizing the navigational primitives (e.g. `<next>` or `<up>`). For that purpose, a method returns the concatenation of Java instructions that sequentially build a corresponding tree. In the example specification (Appendix A), there is only one `<theme>` (01) visible. The `<template>` (07) defines a common layout for the instances to create from the database entries, which are returned by the SQL statement (04–05) within this theme's `<extension>` (02). The default navigation enables the user to switch from one instance to another using either the `<next-button>` (40) or the `<previous-button>` (38).

Compilation is finally accomplished by existing tools that are integrated in the GASSI system via facilities for their invocation on all generated files in the target directory and input/output redirection.

6 The Test Assistant

TASSI serves the analytical quality assurance within the EPK-fix development process. With TASSI all those quality features are examined, which can not constructively be ensured by the other EPK-fix tools. TASSI especially inspects the features that are important for an EPC user, i.e. functionality, robustness and usability of the catalogue.

TASSI should enable a tester to systematically and mostly automatically scrutinize *static* and *dynamic* aspects of an EPC without the help of other tools. A strong support for *manual* tests is given. TASSI includes the following features: fully graphical user interface, declarative specification of general and catalogue-specific automatic tests via rules, automatic execution of static tests of all media-objects, support of dynamic tests by a test agent (automatic navigation of the EPC to untested objects, automatic execution of static tests of dynamic objects), error classification via browser, context-sensitive specification and requirements presentation, capture and replay of manual test input, complete test state administration, and automatic test document generation.

In more detail, firstly the EpkMl instance is fed into an automatic *static analysis* which aims at a rule-based static test of each EpkMl element and each used database object. To determine most used database objects and at least one used set of attributes and elements of most EpkMl elements a dynamic model of the EpkMl instance is automatically generated. This model only excludes outcomes of applets and text inputs.

In the *Tourist Catalogue Example* all database objects of the shown page, i.e. one video reference (`15`), one image reference (`12`), eight texts (`11`, `29`), and seven link (`31`) destinations, are determined. The type of a database object is obtainable from the surrounding EpkMl element at its usage point. Type-specific tests are carried out on all atomic EpkMl elements, e.g. if "To other cities" (`39`) is spelled correctly or if `$cities.video$` (`15`) references an AVI video. Top-level grouping EpkMl elements, i.e. pages and windows (`08`), are examined for group errors each time the container contents changes, i.e. in the example anytime a new city page is entered. Group errors are those errors which only occur if some elements are used simultaneously. In the example for instance it could be verified, that the text colours are in adequate contrast to the background colours or that not too many different fonts are used on the page. Each error found is recorded. Each tested EpkMl element and database object is marked.

In a second phase the actual EPC is *dynamically tested*. The aim is to test the execution of each branch of the EpkMl instance, to inspect manually all media-objects and to validate automatically media-objects and elements that have not been recognized in the static analysis. The EPC is navigated to untested parts by a test agent. A tester essentially only acknowledges the end of manual testing of a set of elements, enters errors into a form and requests the navigation to the next element set. In the case of text input and applet calls, the tester is responsible for choosing sensible input for systematic testing.

The test agent tries to carry out all tests of an element set before navigating to the next one and also attempts to follow intuitive navigation paths. In the example, first the city page (`07`) is shown. After the execution of the video (`15`) the tester is asked to confirm the end of manual testing. Then clicking one of the buttons which lead to untested parts is simulated. If there was no such material, the quit button would be chosen. The tester can always steer the test agent to places of her choice.

7 Conclusions and Further Steps

We support the *whole life cycle of the development process of* EPC*s* by developing an integrated package of tools for the production of electronic product catalogues. These tools aim at the efficient production of low cost electronic product catalogues. Through the features and services observed in current EPCs [6], we identified the components and characteristics of the language EpkMl and the features of each assistant. The *catalogue developer* is assisted from the beginning of the first interview by special editors and a presentation assistant. During the catalogue design and generation phase the assistance is realized with other editors and class libraries. Tests accompany the entire development process including the final EPC.

In our approach, an EPC is the result of cooperating experts working at various places on different aspects of the catalogue. In order to keep track of the correct versions of the documents and programs produced so far, version management is required. We are currently implementing a WWW based project server, which allows to define and manage users and projects with appropriate access capabilities. Furthermore, the server manages revisions of documents and identifies official releases of software modules.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers—Principles, Techniques, and Tools.* World Student Series. Addison-Wesley, 12 edition, 1995.
2. B. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, May 1988.
3. I. Dilli and H.-J. Hoffmann. DIADES II: A Multi-Agent User Interface Design Approach with an Integrated Assessment Component. SIGCHI bulletin, ACM Press, April 1991.
4. C. Goldfarb. *The SGML Handbook.* Clarendon Press, Oxford, 1994.
5. A. Knapp, N. Koch, and L. Mandel. The Language EPKML. Technical report 9605, LMU München, November 1996.
6. N. Koch and L. Mandel. Catalogues on CD-ROM: The State of the Art. Technical report 9610, Ludwig–Maximilians–Universität München, December 1996.
7. M. Reiser. *The Oberon System—User Guide and Programmer's Manual.* Addison Wesley, 1991.
8. A. Turk and H. Stoyan. Erfassung, Verarbeitung und Dokumentation natürlichsprachlicher Äußerungen in der Anforderungsanalyse. In E. Ortner, B. Schienmann, and H. Thoma, editors, *Natürlichsprachlicher Entwurf von Informationssystemen*, pages 32–46. Universitätsverlag Konstanz, May 1996.
9. A. Wasserman and P. Pircher. A Graphical, Extensible Integrated Environment for Software Development. *ACM SIGPLAN Notices 22:1*, 1987.

A Example

In this section we present the EpkML specification of the tourist catalogue page. Note, that some detailed layout information is omitted.


```
<theme name="general">
 <extension result="cities">
  <sql>
   select name,vid,img
   from tourist
  </sql>
  <template name="city-page">
   <window name="city-window"
           style="bavaria">
    <set name=city value=
      "$cities.name$">
    <frame name="header">
     <p>$city$
     <img src="$cities.img$">
    </frame>
    <frame name="left-col">
     <video src=
      "$cities.vid$">
      <stop-button>
     </video>
    </frame>
    <frame name="right-col">
     <sql result="out">
      select issue subtheme
      from tourist
      where theme='$city$'
     </sql>
     <itemize>
      <make-items from="out">
       <frame>
        <button>
         $out.issue$
         <on-click>
          <open name=
            "$out.subtheme$">
         </on-click>
        </button>
       </frame>
      </make-items>
     </itemize>
    </frame>
    <previous-button>
    <p>To other cities
    <next-button>
    <button name="to-reg"
            style="city">
     <img src=
       "icons/reg.jpg">
     <on-click>
      <open name="reg-form">
     </on-click>
    </button>
    ...
   </window>
  </template>
 </extension>
</theme>
<main>
 <open name="general">
</main>
```