VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Anzhela Dobrovolskaia

DITA-OT Plugin Customization

Case study: DITA to Qt Help

VAASAN AMMATTIKORKEAKOULU
Degree Programme of Information Technology

TIIVISTELMÄ

Tekijä	Anzhela Dobrovolskaia
Opinnäytetyön nimi	DITA-OT asiakaskohtainen toteutus: DITA dokumentaatio Qt –ohjeeksi.
Vuosi	2017
Kieli	Englanti
Sivumäärä	68
Ohjaaja	Timo Kankaanpää

Nopeasti kehittyvän teknologian maailmassa tarkasti järjestelty dokumentaatio tuli tärkeäksi osaksi jokaista onnistunutta projektia. Tekninen dokumentaatio sisältää yleensä kolmenlaista tietoa: käsittelyohjeita, taustatietoa ja viitteitä. Epätarkasti luodun dokumentaation yleinen piirre on sisällön sekarakenne. Tämä voi vaikeuttaa luettavuutta ja johtaa käyttäjien sekaannuksiin.

DITA (Darwin Information Typing Architecture) ratkaisee ongelman ottamalla käyttöön tiukan kirjoitusmallin. Luodessaan luvun kirjoittajan on noudatettava seuraavia sääntöjä: se on keskittynyt yhteen ideaan, ja kuulu yhteen tarjotuista tietotyypeistä tyyppiin: konsepti, tehtävä ja viite. DITA tarjoaa XML-pohjaiset mallit kaikille näille kolmelle tietotyypille.

DITA-OT (DITA Open Toolkit) on DITA-lähteen käsittelytyökalu. DITAlla kirjotettua informaatiota voidaan muuntaa PDF:ksi, HTML:ksi, Web Helpksi ja muiksi tiedostomuodoiksi. Apache Ant script, XSLT, ja Java ovat ydinkielet useimmissa DITA-OT:n pricessointimoduuleissa.

Tämän projektin tavoitteena oli kehittää DITA-OT-muunnosskenaario, joka tuottaa DITA-sisällöstä Qt-ohjetiedostot Qt Help Framework -ohjelmalle. Skenaario toteutettiin luomalla DITA-OT -laajennus, joka käyttää sisäänrakennettuja muunnoksia "DITAsta XHTML:ään" ja "Yhdistä DITA-sisältöä" sisääntulona Qt-tiedostojen luomiseen. XHTML-muunnos valittiin paremmaksi ratkaisuksi kuin HTML5, koska se oli paremmin yhteensopiva XML-pohjaisten Qt-ohjetiedostojen kanssa. Laajennus kirjoitettiin Apache Ant script ja XSLT-skenaarioiden avulla.

Avainsanat DITA, DITA-OT, XML, Apache Ant, XSLT

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme of Information Technology

ABSTRACT

Author	Anzhela Dobrovolskaia
Title	DITA-OT Plugin Customization. Case study: DITA to Qt Help.
Year	2017
Language	English
Pages	68
Name of Supervisor	Timo Kankaanpää

In a world of rapidly developing technologies accurately structured documentation has become an important part of every successful project. Technical documentation usually includes three types of information: processing instructions, background information, and references. A common attribute of inaccurately written documentation is a mixed structure of the content. That can cause reading inefficiency and confusion for the users.

DITA (Darwin Information Typing Architecture) solves the issue by introducing a strict model of authoring. Every piece of created information shall follow the rules: it shall be focused on a single idea, and belong to one of the offered information types: concept, task or reference. DITA offers XML-based templates for each of these three information types.

DITA-OT (DITA Open Toolkit) is a software tool for processing of DITA source. Information created in DITA can be transformed into PDF, HTML, Web Help and other formats. Apache Ant script, XSLT, and Java are the core languages for the most of the DITA-OT processing modules.

The goal of this project was to develop a DITA-OT transformation scenario that generates Qt help files for Qt Help Framework from the DITA content. The scenario was implemented by creating a DITA-OT plug-in that uses the "DITA to XHTML", and "Merge DITA content" built-in transformations as an input for generating Qt files. The XHTML transformation was chosen as a better solution than HTML5 due to better compatibility with the XML-based Qt help files. The plug-in was written using Apache Ant script and XSLT scenarios to generate the requested output.

Keywords DITA, DITA-OT, XML, Apache Ant, XSLT

CONTENTS

LIST OF FIGURES, TABLES, AND CODE SNIPPETS

LIST OF ABBREVIATIONS

ANT	Another Neat Tool
DITA	Darwin Information Typing Architecture
DITA-OT	DITA Open Toolkit
Help ID	Unique identifier found in every page of documentation content
HTML	HyperText Markup Language.
UNIC	An embedded engine control system.
UNITool	Maintenance tool for downloading, tuning, monitoring, testing and troubleshooting module software in UNIC.
WADE	Wärtsilä Application Documentation Environment
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language.
XSLT	Extensible Stylesheet Language: Transformation.

1 INTRODUCTION

1.1 Project Background

The thesis was done based on the project request defined by Wärtsilä (hereafter referred to as “the Customer” or “the Client”). Wärtsilä is a corporation known for delivering complete lifecycle solutions for the marine and energy markets, specializing on large combustion engines development.

Today, documentation is a very important element of any engineering project development process. It simplifies the usage and maintenance of a product, accelerates further development implementations and increases the quality of work performance. Every IT company needs to choose in what format to store documentation and how to access or perform the content to users and developers.

During the last few years Wärtsilä has been moving towards a unified structure of documentation for its engine control system. The choice for material storing format was made in favor of DITA standard.

The case study of this thesis documentation is used as a help material in Qt-based application – UNITool. The help source will be displayed in a separate from the main application window browser and will give users access to the supporting documentation. The documentation for a user is sorted based on the following parameters: a UNITool user profile, engine control packages uploaded to the application, an engine type, and a platform.

UNITool is a Wärtsilä maintenance tool that was developed for downloading, monitoring, tuning, testing and troubleshooting embedded module software in the engine control and monitoring system named UNIC. Most of the documentation created by the software developers is related to UNIC Application Software. The application is written in C++ using Qt framework. Qt Help Framework was chosen for displaying supporting documentation for the users.

Why is Qt Help Framework?

As the whole UNITool application is written in C++ using Qt framework, the easiest implementation of the Help function is to use the same language base source. The most important requirements for a help presenting tool defined by the customer were: context sensitivity, index search support and support for dynamic documentation. Qt Help Framework provides all the needed functionality as well as freedom to choose the help content accessing method. In this case UNITool uses QHelpEngine API that embeds help content directly in the application.

Why is DITA?

DITA stands for *Darwin Information Typing Architecture*. DITA is an open standard XML-based architecture for representing documents; it allows storing all documentation in one format without losing publishing performance thanks to DITA Open Toolkit. Moreover, the DITA standard has a long list of features including content reusability and modularity, wide range of output format, continuous development of the standard by OASIS, and availability of fully authorized supporting tools. Additionally, one of the base features of DITA is the use of special information type in the topic form.

Why is DITA-OT?

DITA Open Toolkit is one of the main tools for DITA users. DITA-OT is an open source tool used to transform DITA content to various output formats, for instance HTML, PDF, Eclipse Help, HTML Help, Java Help and other. Created by IBM at the same time as the DITA standard, DITA-OT supports the majority of the features for each version of OASIS DITA specification, including 1.0, 1.1, 1.2, and 1.3. The main reason to choose the DITA-OT as the main tool for maintaining the DITA publishing method is an easy customization process for any of available transformation methods.

1.2 Project Description

The main goal of the thesis is to create a DITA-OT plugin for transforming DITA content into Qt help files that can be used as the source for displaying help content in UNITool application.

The concomitant target was to analyze the possible solution for integrating the help system into the application by comparing the help tool’s functionality to the given requirements.

The requirements for the help tool are:

- Dynamic help support
- Index-based search
- User access level filter
- Context sensitivity

The result of the thesis will only be a part of a bigger project to customize the help source processing and publication, including further development of UNITool Help plugin, database design, file repositories design and other corresponding tasks.

1.3 Author’s role

A similar proof of concept was already implemented for the Customer in the past, but after software updates, the previous version became obsolete and stopped working. My target was to find the reason of transformation failure and reconstruct the old solution. After analyzing the provided source, the request for the thesis work was defined. It was stated that the current solution design does not match plugin customization rules established by DITA-OT and needs to be fully redone. The plug-in will be a part of the Wärtsilä Application Documentation Environment (hereafter WADE). WADE is the set of tools used for creating and maintaining documentation.

2 TECHNOLOGIES

2.1 XML

XML abbreviation stands for Extensible Markup Language. XML was developed as a subset of SGML and it is a language with simple, well-formed, and flexible syntax that can be readable by both machines and humans. XML is defined by several free open standards, first of all, by World Wide Web Consortium and several other related specifications. XML documents describe a class of data objects, and, to a limited extent, the behavior of computer programs that process the documents.

“XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. Markup takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, and processing instructions /5/.”

2.2 Apache Ant

Apache Ant, ANT stands for Another Neat Tool, and it is a Java library and a command line tool. An Ant script contains targets and extension points dependent upon each other that automate software build processes. Apache Ant is an XML-based script, which was developed as an open standard. It was first promoted as an independent project in the year 2000.

2.3 XSLT

“XSLT, which stands for eXtensible Stylesheet Language: Transformation, is a language which, according to the very first sentence in the specification […], is primarily designed for transformation at one XML document into another /6/.”

XSLT is not bound to one type of document that it is able to generate. By applying different technics, developers can convert input XML source into plain text, HTML, or to XSL Formatting Objects that later can be used for generating PDF, PNG and others.

XSLT is a continuously developing language. The first specification was defined on 16th of November, 1999, and the latest revision of the XSLT specification, version 3.0, was released on 8th of June, 2017.

2.4 Qt Help Framework

Qt help framework is a set of tools for generating and viewing Qt help files. HTML documents with help source information, table of contents, and index keywords that are stored in a compressed help files which are integrated into applications as Qt compressed help and Qt help collection.

Generation of the Qt help requires two more files to interact with the help system – Qt help project and Qt help collection project. Compiling of Qt compressed help and Qt help collection is done by qcollectiongenerator, a tool provided with the Qt installation pack. Figure 1. presents the schema of Qt documents collaboration.

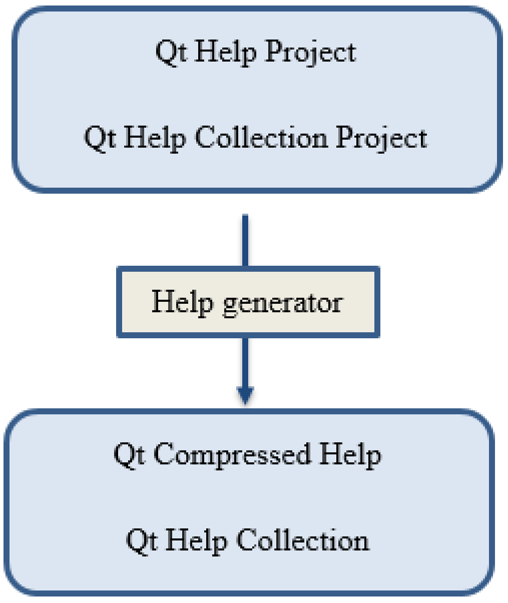


Figure 1. Qt help files collaboration.

Qt Help Project is an input file for the help generator. It contains information about table of contents specializations, indices, and references to all documentation content. Qt Help project is an XML-based file with .qhp extension.

Qt Compressed Help is an output file of the help generator. It is a binary file with .qch extension. The file contains information collected from the Qt Help Project and the compressed documentation files.

Qt Help Collection Project acts also as the input file for the help generator. Qt Help Collection Project contains customization parameters for Qt Assistant, and references to the Qt compressed help files to include in the help collection. It is an XML-based file with .qhcp extension.

Qt Help Collection is an output file of the help generator. It contains information about the included compressed help files as well as custom information, such as filters. It is a binary file with .qhc extension.

2.5 Qt Assistant

“Qt Assistant is a tool for viewing on-line documentation in Qt help file format /7/.” It operates on the binary files created with the help generator.

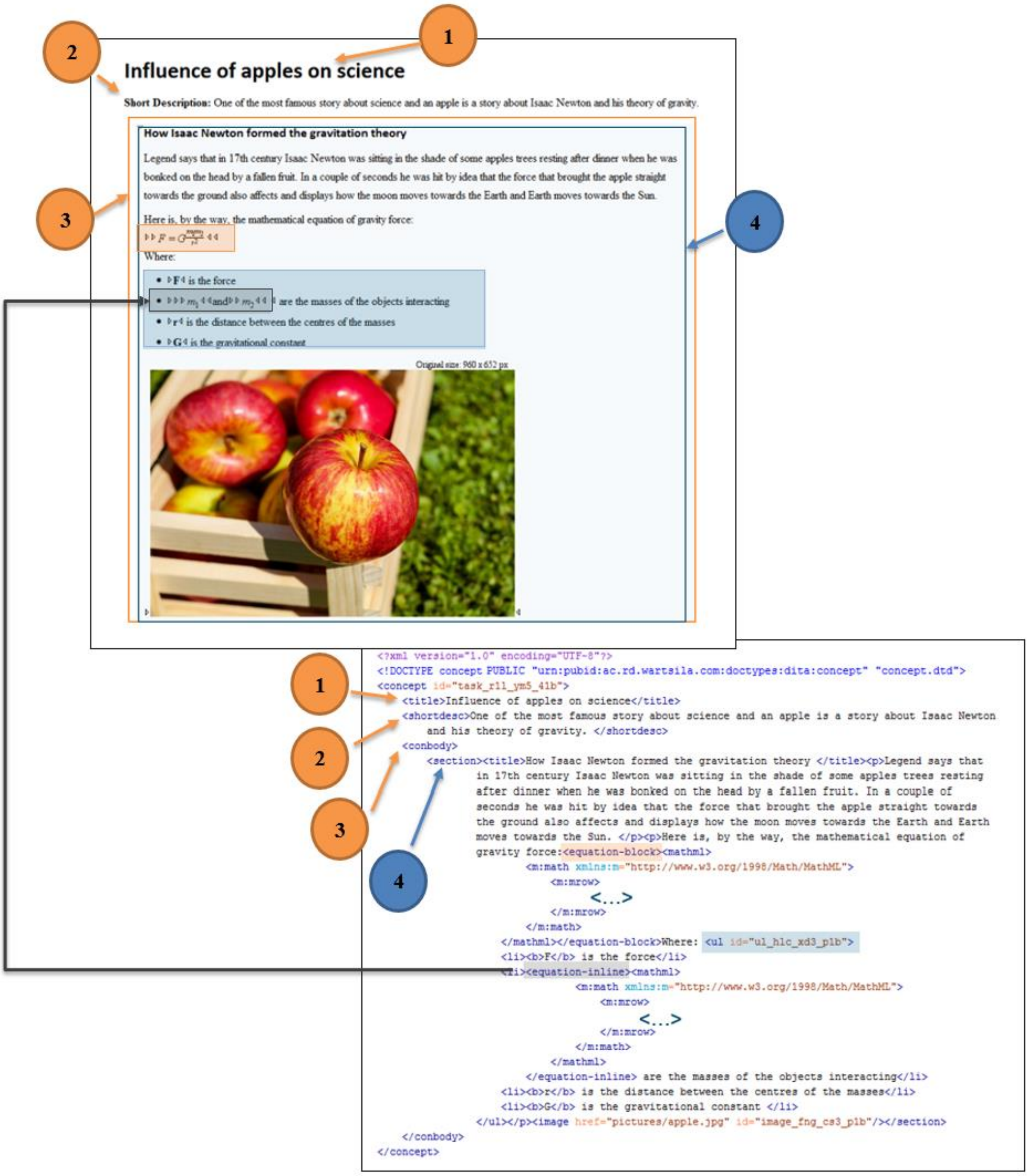
3 DITA

"The Darwin Information Typing Architecture (DITA) is an OASIS Standard that defines an XML architecture for designing, authoring, publishing, and managing content. Content that was developed using the DITA (pronounced dita – uh) model can be easily published to print, PDF, the web, help systems, and other deliverables, depending upon the needs of the users /1/."

To understand DITA, the idea of the topic-based authoring needs to be described and investigated first. The topic-based writing is based on minimalism. The minimalist approach to information design emphases creating short and focused on a single idea content structured based on an information type. There are three main topic types in DITA offered to a user to compose the information: concept, task and reference. Each of these three information types contains a basic set of content units, presented as XML elements, which encompass all the essential needs for technical documentation authors and readers.

However, DITA is not just a set of XML-based tools or out-of-the-box schemas or Document Type Definitions (DTD) that gives an author a possibility to immediately start designing technical documentation with well-designed XML structure. DITA provides an open source technology with an active support team, and continuously developing process of the DITA specification and tools.

The open source technology provides a wide range of development possibilities for DITA authors. The information developers are able to customize existing or create completely new rules for processing content in case a requested information content does not correspond to the prebuild structure of the standard set of information types. The default DITA standard includes more than four hundred elements. Those elements contain attributes, and by using the DITA specialization techniques, writers can rename, remove or add standard XML elements used in DITA depending on the writing purpose. The following Figure 2. visualizes the DITA XML elements, as can be seen, most of the tags name can be defined easily, for instance, the most used elements in DITA are <title> for titles, <shortdesc> for short description, <ul> - for unordered lists, <p> for paragraphs and many others.



Influence of apples on science

Short Description: One of the most famous story about science and an apple is a story about Isaac Newton and his theory of gravity.

How Isaac Newton formed the gravitation theory

Legend says that in 17th century Isaac Newton was sitting in the shade of some apples trees resting after dinner when he was bonked on the head by a fallen fruit. In a couple of seconds he was hit by idea that the force that brought the apple straight towards the ground also affects and displays how the moon moves towards the Earth and Earth moves towards the Sun.

Here is, by the way, the mathematical equation of gravity force:

$F = G\frac{m_1 m_2}{r^2}$

Where:

- **F** is the force
- m_1 and m_2 are the masses of the objects interacting
- **r** is the distance between the centres of the masses
- **G** is the gravitational constant

Original size: 960 x 652 px

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concept PUBLIC "urn:pubid:ac.rd.wartsila.com:doctypes:dita:concept" "concept.dtd">
<concept id="task_rll_ym5_41b">
    <title>Influence of apples on science</title>
    <shortdesc>One of the most famous story about science and an apple is a story about Isaac Newton
        and his theory of gravity. </shortdesc>
    <conbody>
        <section><title>How Isaac Newton formed the gravitation theory </title><p>Legend says that
                in 17th century Isaac Newton was sitting in the shade of some apples trees resting
                after dinner when he was bonked on the head by a fallen fruit. In a couple of
                seconds he was hit by idea that the force that brought the apple straight towards
                the ground also affects and displays how the moon moves towards the Earth and Earth
                moves towards the Sun. </p><p>Here is, by the way, the mathematical equation of
                gravity force:<equation-block><mathml>
                        <m:math xmlns:m="http://www.w3.org/1998/Math/MathML">
                            <m:mrow>
                                <...>
                            </m:mrow>
                        </m:math>
                    </mathml></equation-block>Where: <ul id="ul_hlc_xd3_plb">
                    <li><b>F</b> is the force</li>
                    <li><equation-inline><mathml>
                                <m:math xmlns:m="http://www.w3.org/1998/Math/MathML">
                                    <m:mrow>
                                        <...>
                                    </m:mrow>
                                </m:math>
                            </mathml>
                        </equation-inline> are the masses of the objects interacting</li>
                    <li><b>r</b> is the distance between the centres of the masses</li>
                    <li><b>G</b> is the gravitational constant </li>
                </ul></p><image href="pictures/apple.jpg" id="image_fng_cs3_plb"/></section>
    </conbody>
</concept>
```

Figure 2. DITA elements in a common document view.

Additionally, DITA provides conditional processing capabilities to specify elements of content that are required to be included or excluded from an output. That feature simplifies the process of developing information made for different users but covering the same topic. For instance, if a technical documentation describing a software product needs to have different functionality specifications based on the product version, DITA allows writer to accommodate the difference of requesting elements by labelling them with the special attributes.

Finally, the main feature of DITA is the reuse ability. The cornerstones of the DITA authoring are the XML elements that are used for identifying content units in a topic. Collating the usage of basic paragraphs for process descriptions to short description and concepts, step results and reference information elicits not only a better understanding of technical writing both for the author and the readers, but also a wide range of reusable components for further use.

3.1 Topics

"What is a topic other than a conversational piece? In technical information, a *topic*, which is sometimes called an article, has a title and some content. A topic has just enough content to make sense by itself, but not so much content that it covers more than one procedure, one concept, or one type of reference information /2/".

Users who read manuals for different technologies usually do not need to go through full text provided by documentation developers, but readers need to find the answer to one specific question. Therefore, authors shall compose information that can provide answers on specific questions discretely without necessity of reading across large amount of interconnected content.

Since the twenty-first century, topic-based writing has become the standard for well-formed support documentation authoring. Information architects acknowledge the influence of designing consistently structured topics on readability and on information access for users in more linear content structure. Readers are able to simply navigate themselves in technical content by observing the style of essential content units like tasks, background information and references as the topics types are defined by a specific content style design and recommended location in a table of content.

Each DITA information type is based on a common structure – the DITA topic. The DITA topic defines a starting point for the information type specialization. The base topic structure consists of required and optional components specified in the table below. (Table 1.)

Table 1. Base structure of a DITA topic.

Component	**Description**	**Specification**
Topic Element	The root element of the DITA topic with an "id" attribute.	Required
Title	Specifies the subject of the topic.	Required
Short Description or Abstract	Briefly describes the basic idea of the topic.	Optional
Prolog and metadata	Prolog and metadata categorize, summarize, and label a topic. Specify audience, keywords and index terms, the topic authors, the product with which the topic is associated, the hardware or software platform, and date related information.	Optional
Body	Contains the topic content based on an information type.	Optional
Related Links	Reference to supporting information.	Optional

By writing content in separate topics, creating connections between related topics, and then organizing them into logical groups, information authors can construct a coherent web of information that is convenient for navigation, understanding, and consumption. Created topic collections can be published as PDF, websites, online help systems, and others. An example of linked and organized web of information presented as concept, reference and task topics is shown in Figure 3.

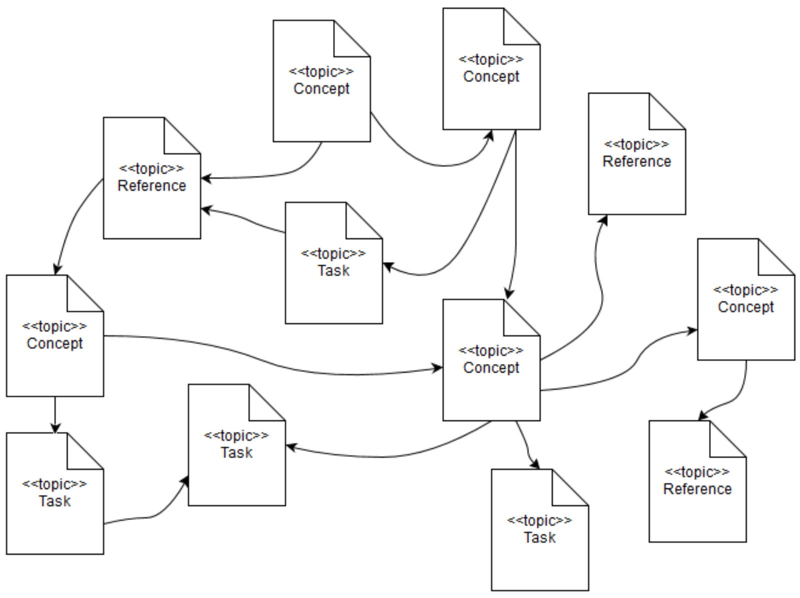


Figure 3. Linked and organized topics that form a web of information.

Technical documentation mostly contains at least three types of information: background or conceptual information, technics and operations, and short reference information. The main-stay of the DITA authoring is separating those types in different topics. To simplify the process of authoring and delivering information that effectively divides content by type and purpose, DITA presents three main topic types: a concept, a reference, and a task:

- A concept topic provides the essential information about a product, a process or a task.
- A reference topic documents one type of reference information related to a concept or a task, for example, product specifications, class descriptions, equipment lists etc.
- A task topic provides step-by-step instructions of one procedure.

Each topic type is a separate XML file with built- in DTDs and XLM Schema Definition (XSD) that defines structure of a topic and its elements set.

One of the most common characteristics of ordinarily written technical content is a mixed structure. For instance, a step-by-step procedure description that contains a large amount of descriptive information about a product and a table of reference in the end, or concept information that provides task instructions incorporated in table cells. And even though DITA allows authoring of mixed information types, it is not recommended. A careful and strict separation of the information into the provided DITA types builds dynamic and flexible documentation that will also be beneficial to use for the readers.

3.2 Maps

A DITA map is the core of any technical documentation created with the DITA standard. A DITA map defines connections and relationships among a set of created DITA topics. Writers can define an information path for readers by organizing topics into hierarchies and groups. A DITA map is the XML file that is used for binding created topics together. The extension of the file is .ditamap.

The structure of a DITA map is simple and intuitive. The <map> element is the root element which contains content of the DITA map, then, an element <title>. The <title> exposes idea of the created DITA map and helps writers and readers to understand the purpose of it. The value specified in the title will be displayed in the output in several formats:

- PDF: the cover page of the document
- In web browser, the <title> value will be shown as the basic title in the browser toolbar
- As title of the section in table of contents

Topic references are specified in the element <topicref>. Family type relationships can be defined by nesting topics inside the element. For instance, a topic that contains other topics is called *parent* topic, the nested topics called *child* topics, and siblings are the topics which are located at the same level. Figure 4. shows the example of a DITA map structure.

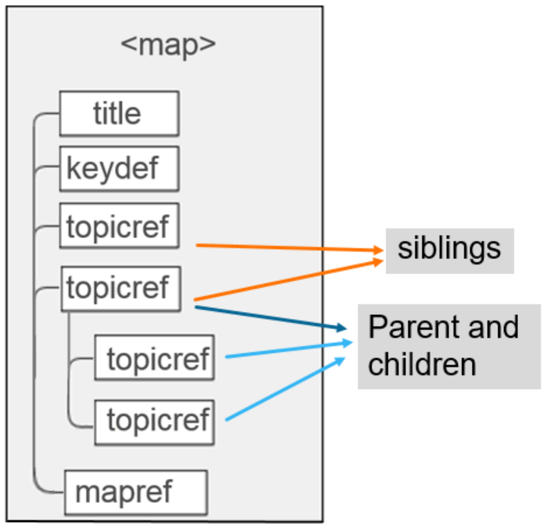


Figure 4. DITA map structure.

An accurate and meaningful structure of a DITA map will help not only the users to follow the information flow smoothly, but also simplify the writer's job by creating some shortcuts for documentation. Shortcuts can be presented as keys with specified values. The reference for the key can be a single word, a phrase or even a complex information container. In a DITA map the element <keydef> is used to define keys for the content. The main advantage of defining keys in a DITA map is visibility in further development of documentation. For example, to change the release number, only the value in the key definition needs to be changed to update all the references to it within the documentation.

In addition, in a DITA map the writers can specify the links among topics. Using the relationship table, authors can manage and store all the related links in an information set.

3.3 Content Reuse

Often in technical documentation writers need to use the same content such as application name and its version number, safety hazard statements or other messages several times or in multiple topics. By introducing @conref (content reference), @keyref (key reference), and @conkeyref (content key reference) attributes, DITA simplifies the authoring, allowing writers to reference reusable content, instead of writing it repeatedly.

Writers need to specify the content units once for being able to access and reference them automatically in multiple topics. DITA content is reused on element level, meaning that if a paragraph is set to be a reusable element, in a new topic it can appear only as a paragraph.

3.4 Where using DITA will be beneficial

Using DITA will be beneficial for any company that is seriously committed to managing information with topic-based and standards-governed approach. DITA brings significant possibilities to technical writers for creating effective and collaboratively written documentation.

Creating information according to the DITA rules for certain information type allows to produce content that can be easily reused across several deliverables. DITA supports conditional processing allowing the creation of different output from a single source by filtering the content based on the writer’s needs.

DITA provides a possibility to specialize any of its processes to fit the requirements of almost any organization. The specialization is available not only for the actual DITA structure and rule but also for adding new types of elements.

Writing information in DITA reduces the cost of localization and translation. DITA topics are based on XML and can be transformed to various types of formats and once they are written and approved they can be transformed to a source required by a translation memory system. Moreover, each DITA topic is a separate file, therefore only new or revised topics need to be sent for translation.

A list of authoring possibilities using DITA includes the following abilities for writers

- Easy and quick information reuse across several deliverables
- Agile technic for customization of topic structures and elements

Currently the DITA standard is used in 693 firms around the world, including Wärtsilä, ABB, Citec, Apple, Adobe, Amazon, Boeing, Cisco, Dell, HTC and many others. The majority of the companies are working in information technology, telecommunication or machinery segments. The information about DITA usage were collected by the Keith Schengili-Roberts /3/, who is one of the first managers of a DITA documentation team in AMD Company,

4 DITA-OT

4.1 What is DITA-OT

DITA Open Toolkit, in short DITA-OT, is a software tool for processing DITA source. DITA-OT contains standard grammar definition files defined by OASIS, and uses Ant, XSLT, and Java to publish DITA content into various deliverable formats. It is an implementation of DITA that is usually shipped together with DITA products, for instance, XML editors or content management tools, to give support to information developers with publishing functionality.

"The DITA-OT implements a multi-stage, map-driven architecture to process DITA content. Each stage in the process examines some or all of the content; some stages result in temporary files that are used by later steps, while others stages result in updated copies of the DITA content. Most of the processing takes place in a temporary working directory; the source files themselves are never modified /4/."

DITA-OT contains the instructions for transformation scenarios, which are used for creating different deliverables out of DITA input. All existing scenarios can be separated by output format types: PDF or HTML-based formats. The design of DITA-OT is based on the linear sequence of separate modules. In the pre-processing stage the toolkit uses the same set of modules for all transformations, and later it follows the line that is specific for a requested format.

4.2 Role of ANT, XSLT and Java in DITA-OT

Apache Ant script is the core base language for most of the processing modules. Being easy customizable and extendable, Ant has become the main controller of transformation processes. Meanwhile, within the Ant script some of the steps are implemented in either XSLT or Java.

XSLT is mainly used for setting rules for transforming and modifying DITA topics depending on the required format. It specifies the styling of DITA elements and its processing.

Java is used for processes that could not have been implemented with XSLT, such as steps that involve copying files or using standard Java libraries.

4.3 Pre-processing modules

All DITA Open Toolkit transformation scenarios start with a set of modules that are common for every output format type. Each stage corresponds to an Ant target in the build pipeline.

The main purposes of the pre-processing stage are analyzing an input information and resolving all the internal content references, and filtering instructions. During this stage, DITA-OT creates lists of files based on specific parameters in a temporary directory, copies and filters all the files, adds debugging attributes to every single element, and resolves the metadata. After the pre-processing stage is done, the transformation process is diverged based on the requested output format.

The scenario of the main steps is described in Table 2.

Table 2. The steps of the pre-processing phase.

Name	**Ant target**	**Description**	**Language**
Generate list	gen-list	The first step of the pre-processing stage is the examination of the input files. Based on the analysis, several lists are created based on the file type and the document properties in a temporary directory. For instance, one list contains all topics where content was referenced within `@conref` attribute. Later in the pipeline this list will be used as a reference for resolving and compiling final content of the documents.	Java
Debug and filter	debug-filter	The debug and filter step processes input DITA content and creates copies in a temporary directory. While copying DITA content, debugging information is inserted in each element using the `@xtrf` and `@xtrc` attributes, filtering is performed if a DITAVAL file was specified, and the table column names are adjusted to ensure correct processing of table content in case the content was referenced from another table.	Java

Resolve map references	mapref	The mapref step resolves map references from one DITA map to another. As a result of the current step, the map reference in the main DITA map is replaced by the topics from the other map.	XSLT
Copy related files	copy-files	This step copies non-DITA files referenced in the input DITA source to the output directory.	Java
Resolve keyref	keyref	"The `keyref` step examines all the keys that are defined in the DITA source and resolves the key references. Links that make use of keys are updated so that any @href value is replaced by the appropriate target; key-based text replacement is also performed, and the key definition list file is written to the temporary directory."/8/	Java
Conref push; Conref; and Resolve code references	Conrefpush; conref; and coderef	During these steps, the content that was referenced from one DITA topic or non-XML files to another DITA topic is pulled to a specified location.	Java; Java; XSLT
Move metadata and pull content into maps	move-meta-entries	The move-meta-entries step affects the topics where parameters, such as index entries, copyrights, dates or similar, were defined in the prolog section. During the step, the metadata values are pushed between the maps and the topics setting the processing order of the affected topics. The content from the referenced topics is pulled into the maps, and then within the maps, the data is formed as a cascade.	Java and XSLT
Map-based linking	maplink	During this step, the links are collected based on the map and then moved into the referenced topics. The links are created based on the hierarchy structure of the DITA map, specified attributes and the relationship table.	Java and XSLT

Pull content into topics	topicpull	The topicpull step processes the content that was linked to the DITA documents	XSLT

4.4 Style formatting

The best technic for minimal customizing of existing styles in DITA-OT, is overriding predefined the XSLT scenarios or a CSS files. By creating a single XSLT or CSS file with new instructions, the developer can pass new parameters to the original stylesheet and override it.

This approach is appropriate for small changes. However, for more complicated customization scenarios, it is better to create a completely new plug-in. The plug-in creation process will be described later in the document.

4.5 Filtering techniques

For generating the source for several deliverables based on a single input, information developers shall specify the instructions for the output processor. “A conditional processing profile (DITAVAL file) is used to identify which values are to be used for conditional processing during a particular output, build, or some other purpose. The profile should have an extension of .ditaval/9/.”

The root element for the DITAVAL file is the <val> element. In most of the cases, the root element contains one or more <prop> elements. The <prop> element identifies an attribute to take action on using the @att attribute, the available values are: audience, platform, product, props, and otherprops. If no attribute type is defined, the <prop> element sets a default action for the entire DITAVAL file. The possible actions to be taken for a <prop> element are: include (include the content in output), exclude (exclude the content from output), flag (flag the content in the output, a developer can set a phrase or an image for flagging the element where specified attribute was set), and passthrough (keep the attribute value as a part of the output stream for a later runtime engine processing). The action is defined in the @action attribute. The value for the attribute to be acted on is defined using the @val attribute.

A sample of a DITAVAL file is presented in Code Snippet 1.

```
< val>
   <prop att="audience" val="master" action="exclude" />
</val>
```

Code Snippet 1. Sample DITAVAL file.

In the presented code, the “exclude” action is performed on the element where the attribute @audience is set to “master”. This DITAVAL file instructions can be used, for instance, in a case when the output documentation is requested to be generated for newcomer engineers.

4.6 Customization techniques

There are several methods that can be used for extending or customizing internal DITA-OT processes. Even though it is possible to directly change the toolkit code for customization purposes, DITA-OT developers advise to create a separate plug-in that can accomplish almost any modifications. A separate plug-in is safe from the system updates, it can act independently or be built upon other plug-in, and the transformation scenarios maintained in plug-ins are easy to build more complex, but agile mechanisms.

A plug-in represents a mechanism of handling the process modules after the pre-process stage is completed. The plug-in consists of the repository, and shall be stored under the /plugins directory inside DITA-OT. The instructions mainline is specified in the plugin.xml file that shall contain a unique identifier; it ensures the visibility of the plug-in to the rest of the toolkit.

The folder and the file structure of the new plug-in shall follow a suggested template: in the root folder, only Ant scripts shall be located, while supporting files shall be placed under separate new folders that are named shortly but descriptive.

Additionally, the developers can pass new XSLT parameters into existing XSLT files and add Java libraries to the global DITA-OT clathpath. Clathpass is a variable that contains information about a location of the Java libraries in the user’s computer.

5 SYSTEM DESCRIPTION AND DESIGN

5.1 Requirements specification

The scope of this thesis is to design a transformation scenario that generates Qt help packages from DITA source.

The requirements for the project were defined based on the user stories collected during an analyzation stage as well as based on the previous implementation and the use of DITA to Qt help packages transformation, UNITool application functionality, DITA-OT customization practices, and requests for an additional functionality.

The priority for the requirements were defined as follows:

- **Must have**. Requirements which are essential for successful implementation of the project.
- **Highly recommended**. High priority requirements that should be implemented if possible. Workarounds are available.
- **Nice to have**. Desirable requirements, if not implemented the project will still be accepted.

The following tables contain requirements separated by a type of functionality it covers.

Table 3. Authoring requirements.

ID: R1	Qt Assistant for viewing the transformed UNITool help publication shall be opened automatically after the transformation is completed	**Priority**: Nice to have

Table 4. Publication usability requirements.

ID: R2	All UNITool help transformation related resources shall be moved under UNITool help DITA-OT plugin	**Priority**: Highly recommended

ID: R3	Index keyword generation for UNITool help publication shall follow the general DITA conventions for creating an index	**Priority**: Highly recommended

Table 5. Build automation requirements.

ID: R4	UNITool help publication DITA processing shall support content filtering by UNITool user access level (viewer, operator, expert, developer)	**Priority**: Must have
ID: R5	Default processing mode of UNITool help transformation scenario shall be “strict”	**Priority**: Highly recommended
ID: R6	UNITool help publication DITA processing shall support content filtering by UNITool user profile (developer/service)	**Priority**: Highly recommended
ID: R7	UNITool help publication shall be possible to run using DITA-OT command line tool.	**Priority**: Must have

Table 6. Technical requirements from interface to UNITool help framework system.

ID: R8	The help publication contents shall be designed based on UNITool user access levels (viewer, operator, expert, developer)	**Priority**: Highly recommended
ID: R9	The UNITool help publication shall be passed to UNITool help browser as Qt Compressed Help (.qch) and Qt Help Collection (.qhc) that are generated with the transformation.	**Priority**: Nice to have
ID: R10	Each help publication topic shall have some unique identifier that will be used as Help ID in transformed Qt Compressed Help (.qch) package. Help ID is used in UNITool help framework for activating context sensitivity feature.	**Priority**: Nice to have
ID: R11	UNITool help transformation fails if the DITA source does not meet the DITA-OT's limitations for transforming HTML based outputs	**Priority**: Highly recommended

ID: R12	Authors who wish to transform UNITool help shall install Qt resources additional to WADE	**Priority**: Nice to have
ID: R13	UNITool help DITA-OT plugin shall be based on DITA v1.3	**Priority**: Highly recommended

5.2 Use cases

There are three types of users who interact with the system: “UNITool users”, “Information developers”, and “WADE developers”. The scope of the thesis work bounds the list to “Information developers” and “WADE developers” user groups only. The use case diagrams were designed based on the project scope and the specified requirements.

The preconditions for all the use cases are: the latest WADE version and JRE 7 or later shall be installed. For the use cases 2.0 and later also the latest Qt shall be installed. The transformation can be run on Windows OS.

5.2.1 WADE developers

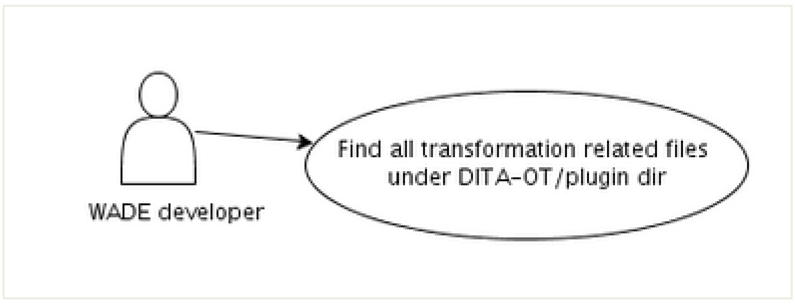

WADE developer
Find all transformation related files under DITA–OT/plugin dir

Figure 5. Use case diagram for the WADE developer user role.

Table 7. Use cases for the WADE developer user role.

Use case: Find DITA to UNITool transformation scenario files under DITA-OT/plugin directory

Description:	All UNITool help transformation related resources can be found under UNITool help DITA-OT plugin, including build and post-processing scripts, xslt-stylesheets, CSS, and image files.
Main Success Scenario:	The location of the files related to DITA to UNITool help transformation is DITA-OT/WADE
Exceptions:	Later plugin customizations add new steps to the processing pipeline that take in use files located outside of DITA-OT/plugin directory
Priority:	High

5.2.2 Information Designers

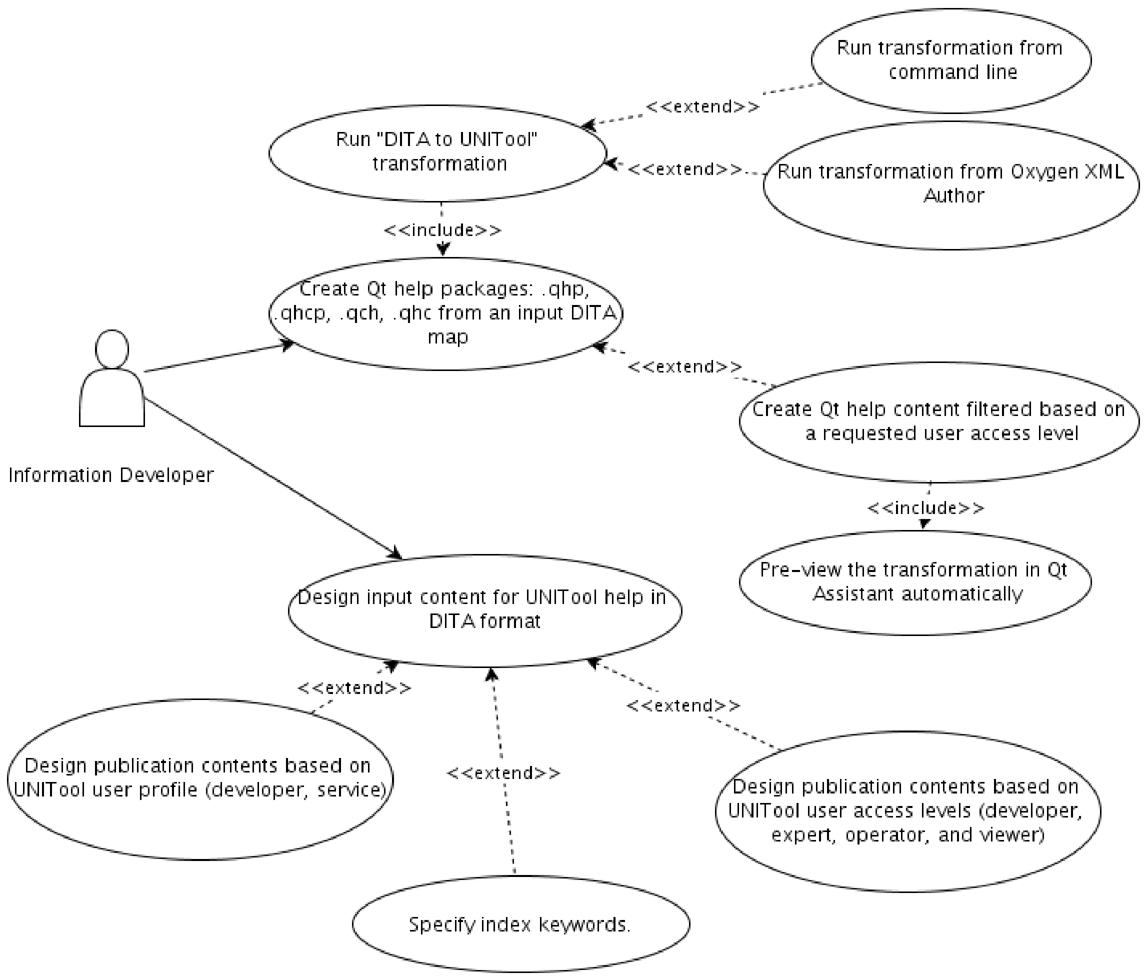

Run transformation from command line
Run "DITA to UNITool" transformation
<<extend>>
<<extend>>
Run transformation from Oxygen XML Author
<<include>>
Create Qt help packages: .qhp, .qhcp, .qch, .qhc from an input DITA map
Information Developer
<<extend>>
Create Qt help content filtered based on a requested user access level
<<include>>
Pre-view the transformation in Qt Assistant automatically
Design input content for UNITool help in DITA format
<<extend>>
<<extend>>
<<extend>>
Design publication contents based on UNITool user profile (developer, service)
Design publication contents based on UNITool user access levels (developer, expert, operator, and viewer)
Specify index keywords.

Figure 6. Use case diagram for the Information Developer user role.

Table 8. Use cases for the Information Developer user role.

Use case 1: Design input content for UNITool Help in DITA 1.3 format	
Description:	The information developer can use features introduced in DITA 1.3 specification to create source for UNITool Help transformation.
Main Success Scenario:	Features introduced in DITA version 1.3 can be used by the user.

Exceptions and errors:	User has outdated WADE version
Priority:	High
Extensions:	Use case 1.1; Use case 1.2.
Use case 1.1: Design publication content based on UNITool user access levels (developer, expert, operator, viewer)	
Description:	The information developer can define user access level by specifying corresponding @audience attribute value to the DITA source.
Main Success Scenario:	Audience attribute has 4 possible values: developer, expert, operator and viewer.
Exceptions and errors:	User has outdated WADE version
Priority:	High
Use case 1.2: Specify index keywords	
Description:	Index keywords are specified within the <index> element. That element is included in DITA specifications.
Main Success Scenario:	The <index> element is a supported element.
Exceptions and errors:	None
Priority:	High

Use case 2: Run DITA to UNITool transformation	
Description:	The information developer is able to start and run DITA to UNITool Help transformation
Main Success Scenario:	The transformation process starts and finishes without failures.
Exceptions and errors:	User has outdated WADE version
Priority:	High
Extensions:	Use case 2.1; Use case 2.2

Use case 2.1: Run transformation from the DITA command line	
Description:	The information developer is able to start DITA to UNITool transformation from DITA command line.
Main Success Scenario:	The transformation process starts and finishes without failures.
Exceptions and errors:	User called incorrect command; the input content is invalid
Priority:	High
Use case 2: Run transformation from Oxygen XML Author	
Description:	The information developer is able to start and run DITA to UNITool Help transformation from Oxygen XML Author.
Main Success Scenario:	The transformation process starts and finishes without failures.
Exceptions:	User has wrong Oxygen XML Author settings; invalid input source
Priority:	High
Use case 2.3: Create Qt help packages: .qhp, .qhcp, .qch, .qhc from input DITA map	
Description:	The user can create Qt help files (qhp, .qhcp, .qch, .qhc) from the input source
Main Success Scenario:	The requested files are created.
Preconditions:	The DITA to UNITool transformation was run on the input source
Exceptions:	User has invalid input source.
Priority:	High
Extensions:	Use case 2.4; Use case 2.5
Use case 2.4: Create Qt help content filtered based on requested user access level	
Description:	Output content is filtered based on the requested access level. The possible values are: developer, expert, operator, viewer, or no filter
Main Success Scenario:	The content is filtered.

Preconditions:	The DITA to UNITool transformation was run on the input source, the access level was specified before the transformation.
Exceptions:	User has invalid input source.
Priority:	High
Use case: Pre-view the transformation in Qt Assistant automatically	
Description:	Created content can be viewed in Qt Assistant that opens automatically after the transformation is completed.
Main Success Scenario:	Qt Assistant is opened, user can see the filtered content. Content is the same as in the input DITA source.
Preconditions:	The DITA to UNITool transformation was run on the input source
Exceptions:	User has invalid input source.
Priority:	High

6 APPLICATION DESIGN

The DITA to Qt help plugin was designed based on the latest recommendations of implementing customized scenarios in DITA-OT.

The basic idea of the transformation is to transform input DITA source into Qt Help packages. The input source shall be provided in the DITA map format. The DITA map shall be structured based on the DITA regulations for creating DITA source for XHTML output. The regulations are:

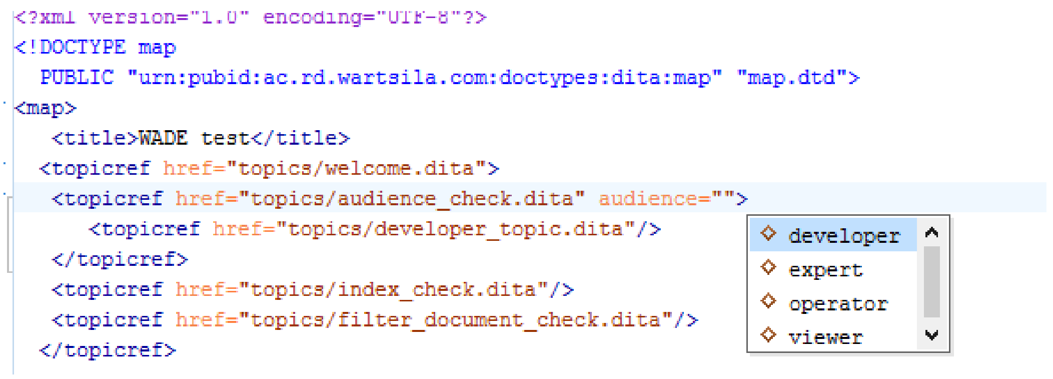
1) Not using <topichead> element, as in <topichead> only a title needs to be specified for placing it in the table of contents, but for consistence of XHTML output, the whole table of content elements shall have a link to the actual source.
2) As the final result in the UNITool Help browser will be a collection of multiple DITA maps, each DITA map shall have a major parent topic that will wrap all the child content.
3) Every DITA map shall have a title.

The documentation needs to be sorted for four different user roles which have different access levels. The roles are: developer, expert, operator, and viewer. The access level is based on “cascade” filtering structure that is represented in Table 9.

Table 9. Description of access level filtering.

Value	Elements included in the output	Elements excluded from the output
developer	developer, expert, operator, viewer	none
expert	expert, operator, viewer	developer
operator	operator, viewer	developer, expert,
viewer	viewer	developer, expert, operator

Documentation developers can specify the user access level by adding an @audience attribute to a filtered element. Writers can use either one of the values or several at once by separating values with a space. The attribute @audience can be applied to most DITA elements, such as <map>, <topicref>, <ul>, <p> and others. If an element does not have any audience specification, the filtering function is not applied. Figure 8. presents an example of using @audience attribute for the <topicref> element in a DITA map which means that the entire DITA topic will be affected by the filtering rules.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map
  PUBLIC "urn:pubid:ac.rd.wartsila.com:doctypes:dita:map" "map.dtd">
<map>
   <title>WADE test</title>
  <topicref href="topics/welcome.dita">
   <topicref href="topics/audience_check.dita" audience="">
      <topicref href="topics/developer_topic.dita"/>
   </topicref>
   <topicref href="topics/index_check.dita"/>
   <topicref href="topics/filter_document_check.dita"/>
  </topicref>
```

Figure 7. Example of applying filter attribute to a DITA topic.

To apply the filters users should specify the user access level before the transformation begins. In Oxygen XML Author it is implemented by showing the dialogue box shown in Figure 9. The dialogue box also includes “no filter” option, meaning that all the input content will be included in the output. No filter option can be used by the writer for testing purposes.

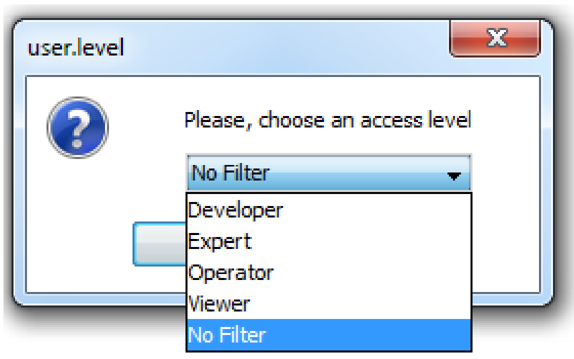
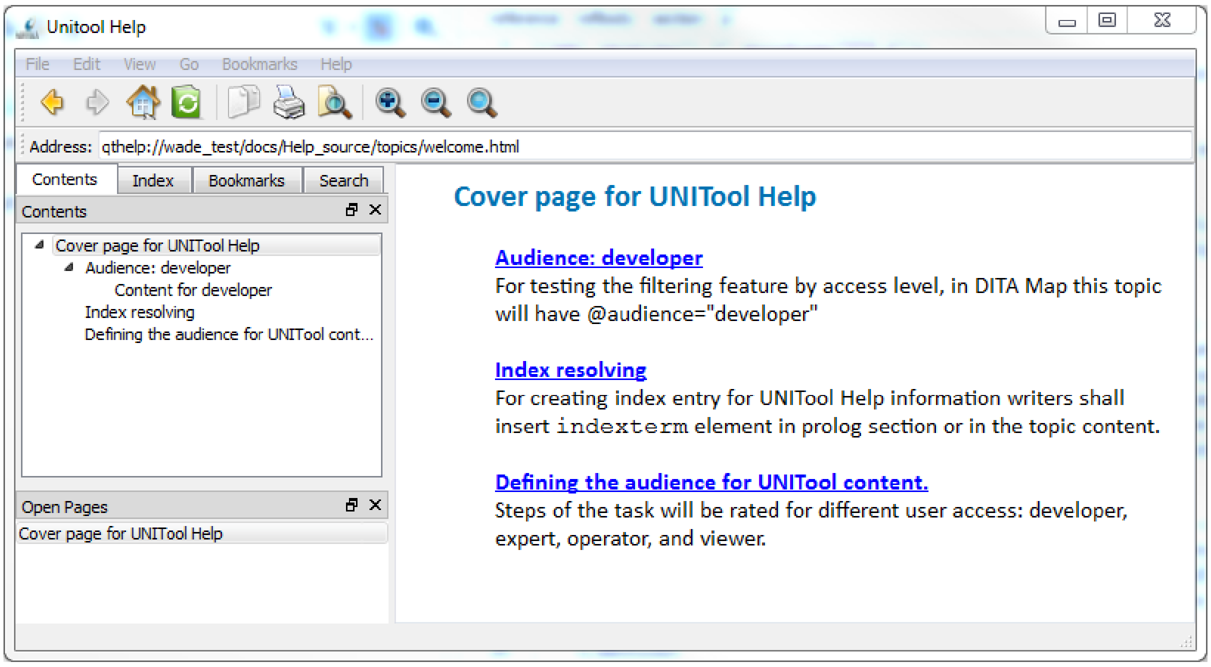


Figure 8. Specifying a user access level in Oxygen XML Author.

To run the transformation from DITA command line, the users should navigate to dita.bat file, which is located at DITA-OT/bin, and call the command: dita -i input.ditamap.dir -f UNITool_Help -Duser.level=developer -o output.dir, where input.ditamap.dir is the absolute path to a DITA map, and output.dir - the output directory path for the generated output.

After the transformation is completed, the generated content can be viewed by the user in Qt Assistant that is opened automatically as the last step of the transformation scenario. Qt Assistant is a software for viewing on-line documentation in Qt help format. Qt Assistant has a similar design and essential functionality to Qt help browser that is used in UNITool. Therefore, the most important features of created documentation can be tested with Qt Assistant before applying the help packages to UNITool.

Figure 10. represents a view in the Qt Assistant with a generated source. In the left section of the window, a reader can see the opened "Content" tab which contain the list of the available information pages. The right side of the window is used for showing the help content.



Address: qthelp://wade_test/docs/Help_source/topics/welcome.html

- Cover page for UNITool Help
  - Audience: developer
    - Content for developer
  - Index resolving
  - Defining the audience for UNITool cont...

Open Pages: Cover page for UNITool Help

Cover page for UNITool Help

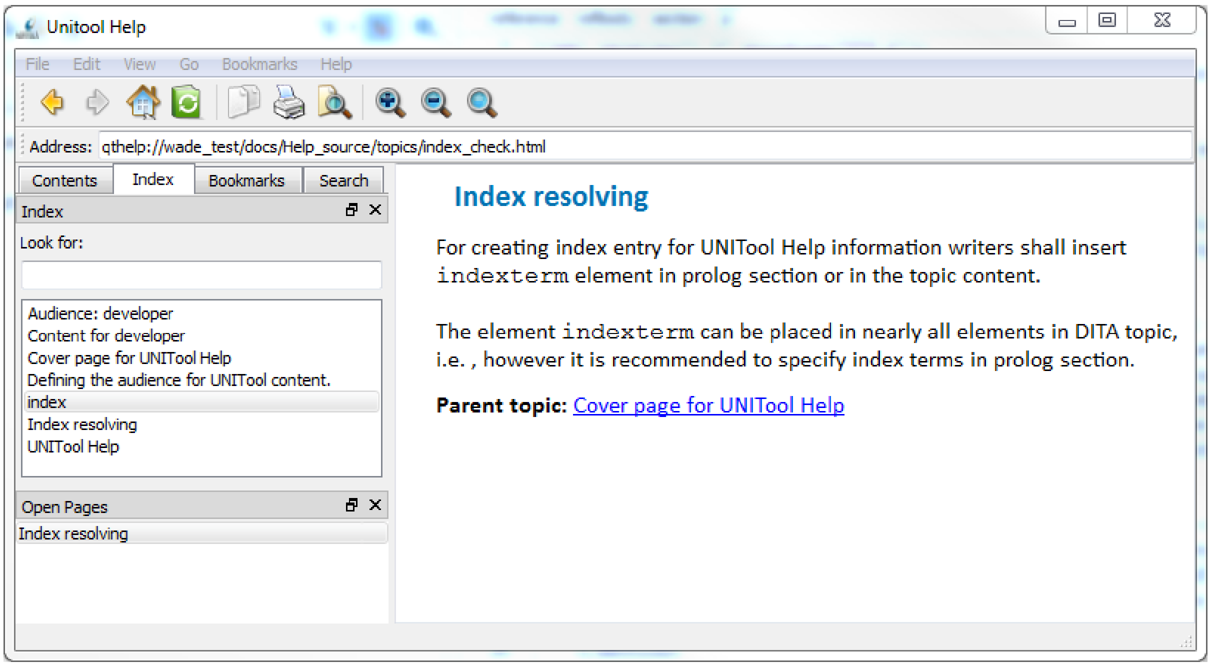
Audience: developer
For testing the filtering feature by access level, in DITA Map this topic will have @audience="developer"

Index resolving
For creating index entry for UNITool Help information writers shall insert indexterm element in prolog section or in the topic content.

Defining the audience for UNITool content.
Steps of the task will be rated for different user access: developer, expert, operator, and viewer.

Figure 9. Qt Assistant with generated source.

The “Index” tab contains the list of words that were collected from the input source. Usually an index entry is a single word or a short phrase that describes the content. The readers will look into the index list when looking for a specific information. After double click on an index word from the list, Qt Assistant refers the reader to the page where the index word was found. Figure 11. shows the opened index tab and the page, where the “index” keyword was found.


Unitool Help
File Edit View Go Bookmarks Help
Address: qthelp://wade_test/docs/Help_source/topics/index_check.html
Contents
Index
Bookmarks
Search
Index
Look for:
Audience: developer
Content for developer
Cover page for UNITool Help
Defining the audience for UNITool content.
index
Index resolving
UNITool Help
Open Pages
Index resolving

Index resolving

For creating index entry for UNITool Help information writers shall insert `indexterm` element in prolog section or in the topic content.

The element `indexterm` can be placed in nearly all elements in DITA topic, i.e. , however it is recommended to specify index terms in prolog section.

Parent topic: Cover page for UNITool Help

Figure 10. Qt Assistant, “Index” tab.

For implementing the feature, the UNITool Help authors should use the <indexterm> or/and the <keyword> elements. In DITA, the index entries and the keywords can be placed almost anywhere in a topic, but most of the index entries should be inserted in the <keywords> element in the <prolog> section.

For UNITool to be able to use Qt help, only 2 files need to be transferred in a specific location. The packages are Qt Help Collection and Qt Compressed Help, the binary files that contain all the help documentation. UNITool requires to place those files under UNITool [version number]/Configuration/Help/[user-role]. The files shall be named QTHelpCollection.qhc and QCH_filename_[user-role].qch.

6.1 Qt help framework architecture

The content for the help documentation is specified in the XML-based files – Qt Help Project and Qt Help Collection project.

Qt Help Project specifies and organizes table of contents, indices and references to the HTML files, the CSS files and additional content. The template of the file is presented in Code Snippet 2. A unique name space for the documentation and a virtual folder must be defined first. The virtual folder will be created to escape absolute links. The name space ensures the uniqueness of the pass. The <filterSection> is the section for defining content related references. The <toc> container is used to specify the table of contents information. HTML source included in documentation can be structured using the <section> element. The <keywords> section is used for specifying the words that will be used as the index keywords, and the <files> section shall contain the list of all the files needed for help generation, including stylesheets, HTML documents, and images.

```
<?xml version="1.0" encoding="UTF-8"?>
<QtHelpProject version="1.0">
  <namespace>test</namespace>
  <virtualFolder>docs</virtualFolder>
  <filterSection>
    <toc>
      <section ref="welcome.html" title="Cover page for Help"/>
    </toc>
    <keywords>
      <keyword name=" welcome" ref=”welcome.html" id="welcome_page"/>
    </keywords>
    <files>
      <file>*.css</file>
      <file>welcome.html</file>
    </files>
  </filterSection>
</QtHelpProject>
```

Code Snippet 2. Source code template for Qt Help Project.

Qt Help Collection Project file contains references to the Qt Compressed Help files, and, in addition, the customization instructions for Qt Assistant design. The structure of the file is presented in Code Snippet 3. The <input> and the <output> elements define the location of Qt

Help Project and name the Qt Help Collection file that will be generated. The <register> container is used for specifying the Qt Collection Help files that will be included in the output.

```
<QHelpCollectionProject version="1.0">
  <!-- Custimizing Qt Assistant view -->
  <assistant>
    <title>Unitool Help</title>
    <startPage>docs/index.html</startPage>
    <applicationIcon>Help_source/wartsila_logo.png</applicationIcon>
    <cacheDirectory>wartsila</cacheDirectory>
  </assistant>

  <docFiles>
    <generate>
      <file>
        <input>ditamap_name.qhp</input>
        <output>QCH_filename_.qch</output>
      </file>
    </generate>
    <register>
      <file>QCH_filename_.qch</file>
    </register>
  </docFiles>
 </QHelpCollectionProject>
```

Code Snippet 3. Qt Help Collection Project template.

The <assistant> container is used for specifying custom parameters for the view of Qt Assistant. The title “UNITool Help” and the Wartsila Corporation icon is defined for the current case.

7 IMPLEMENTATION

The best practice for customizing DITA-OT processes is to create the requested output is designing a separate plug-in under DITA-OT/plugin directory.

The development of the plug-in for the current project started with choosing a unique name for the transformation type. This name will be used for naming the folder, and as a parameter that will be passed to the main build file in DITA-OT. Consequently, it shall be distinctive and special to not overlap with already existing plug-in values. After a careful consideration and investigation of the DITA-OT plugin folder structure, "UNITool_Help" was chosen as the value.

7.1 DITA to XHTML & Merge DITA content plug-ins

Based on the Qt Help framework requirements, the input for creating the Qt help packages needs to be presented as HTML source. Moreover, for activating additional features, for instance, index words search, some additional data shall be collected from the input provided by the user. The "DITA to XHTML" and "Merge DITA content" transformations generates content that is possible to use for the requested purposes.

The output of the "DITA to XHTML" transformation is XHTML pages and a table of contents file. The XHTML transformation can be customized by adding custom CSS files for adjusting the styles in the output webpages, and, additionally, by changing the sequence responsible for generating the table of contents file.

The "Merge DITA content" transformation generates a single XML file out of all the input DITA topics that were referred in a DITA map. The generated file is useful for collecting certain types of information used in the entire document, for instance, key words, terms and other metadata. Originally, the output file is named dynamically based on the name of the input master DITA map, but from this point and later in the text, it will be referred as "merged.xml".

Figure 13. shows the schema of the files involeved in the "DITA to UNITool Help" transformation.

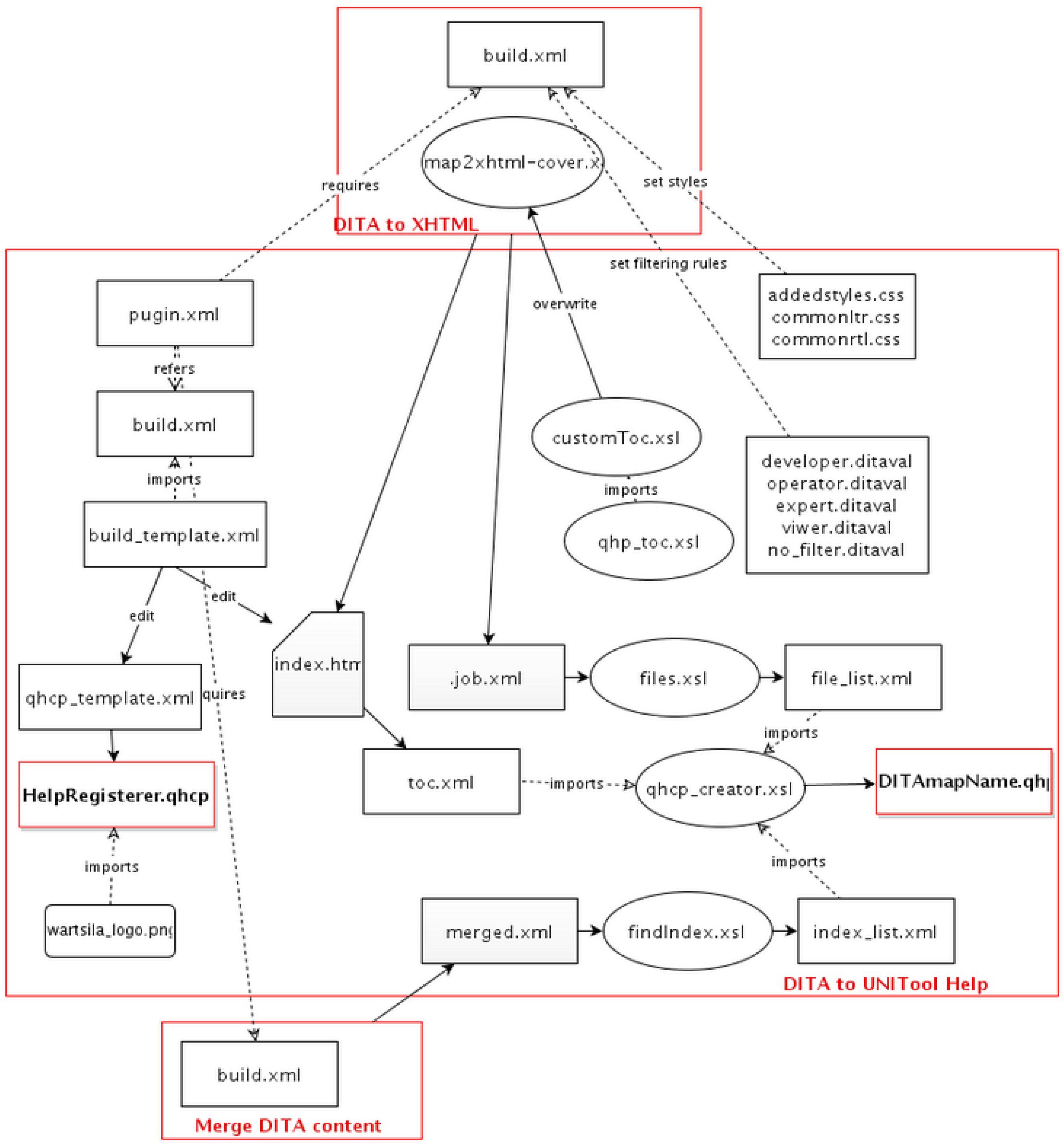


Figure 11. The visual diagram of the files used in “DITA to UNITool Help” transformation.

7.2 UNITool_Help plug-in. Ant script.

The main DITA-OT build files are written with Apache Ant script. For referencing the files from the DITA-OT plugins, the developers can use Ant variables in the form

${dita.plugin.plugin-id.dir}. Ant variables ensure the correctness of the path without relative dependencies.

plugin.xml

A required file for every plug-in is the descriptor file plugin.xml. “The plug-in descriptor file (plugin.xml) controls all aspects of a plug-in, making each extension visible to the rest of the toolkit. The file uses pre-defined extension points to locate changes, and then integrates those changes into the core DITA-OT code /5/.”

Consistence, modularity, possibility to preserve through the toolkit updates, and opportunity to override or develop more complex scenarios without increasing complexity to the extension mechanism are the advantages of creating custom transformations with the plug-ins.

Code Snippet 4. represents the source code of plugin.xml. It is an XML-based file with a root element <plugin>, which has a required attribute – @id. Importantly, the id attribute contains the unique transformation type value – UNITool_Help.

```
<plugin id="UNITool_Help">
   <require plugin="org.dita.xhtml"/>
   <feature extension="dita.conductor.transtype.check" value="UNITool_Help"/>
   <feature extension="dita.transtype.print" value="UNITool_Help"/>
   <feature extension="dita.conductor.target.relative" file="build.xml"/>
</plugin>
```

Code Snippet 4. Source code of plugin.xml.

The <require> element is optional and contains a definition of the plug-in dependencies. For creating Qt Help Packages the input source shall be presented as XHTML files. Therefore, a prebuild plug-in, org.dita.xhtml, was chosen to be the transformation type required for running the UNITool Help plug-in.

The <feature> element supplies values to DITA-OT extension points.

- Dita.conductor.transtype.check adds value “UNITool_Help” to the list of valid transformation type names.
- Dita.transtype.print declares the transformation type as the print type.

- Dita.conductor.target.relative refers to the file with new Ant targets which will be copied to the main build.xml file, allowing new Ant targets to be available to other DITA-OT processes.

build.xml

The content of the build.xml presented in Code Snippet 5. will be copied into the main DITA-OT build.xml file, therefore it contains only a single <import> command that refers to another Ant file - the build_unitool_help.xml. The root <project> element has the name attribute which has the same value as was specified as the unique identifier mentioned in plugin.xml

```
<project name="UNITool_Help">
   <import file="build_unitool_help.xml"/>
</project>
```

Code Snippet 5. Source code of build.xml.

build_unitool_help.xml

The logic of “DITA to UNITool Help” transformation is specified in build_unitool_help.xml. It is an Apache Ant script that is responsible for every step that the “UNITool_Help” transformation does. It customizes the “DITA to XHTML” transformation and takes merged.xml file in use from “Merge DITA content” transformation. The script consists of targets that represent short separate processes which are designed based on a pipeline idea, meaning that the output of one step or several steps is used as the input for the next procedures.

Figure 12. presents the visual diagram of the transformation pipeline. The output of DITA-OT built-in transformation (merged.xml, temporary files, HTML topics, and table of contents files) is processed with the following steps.

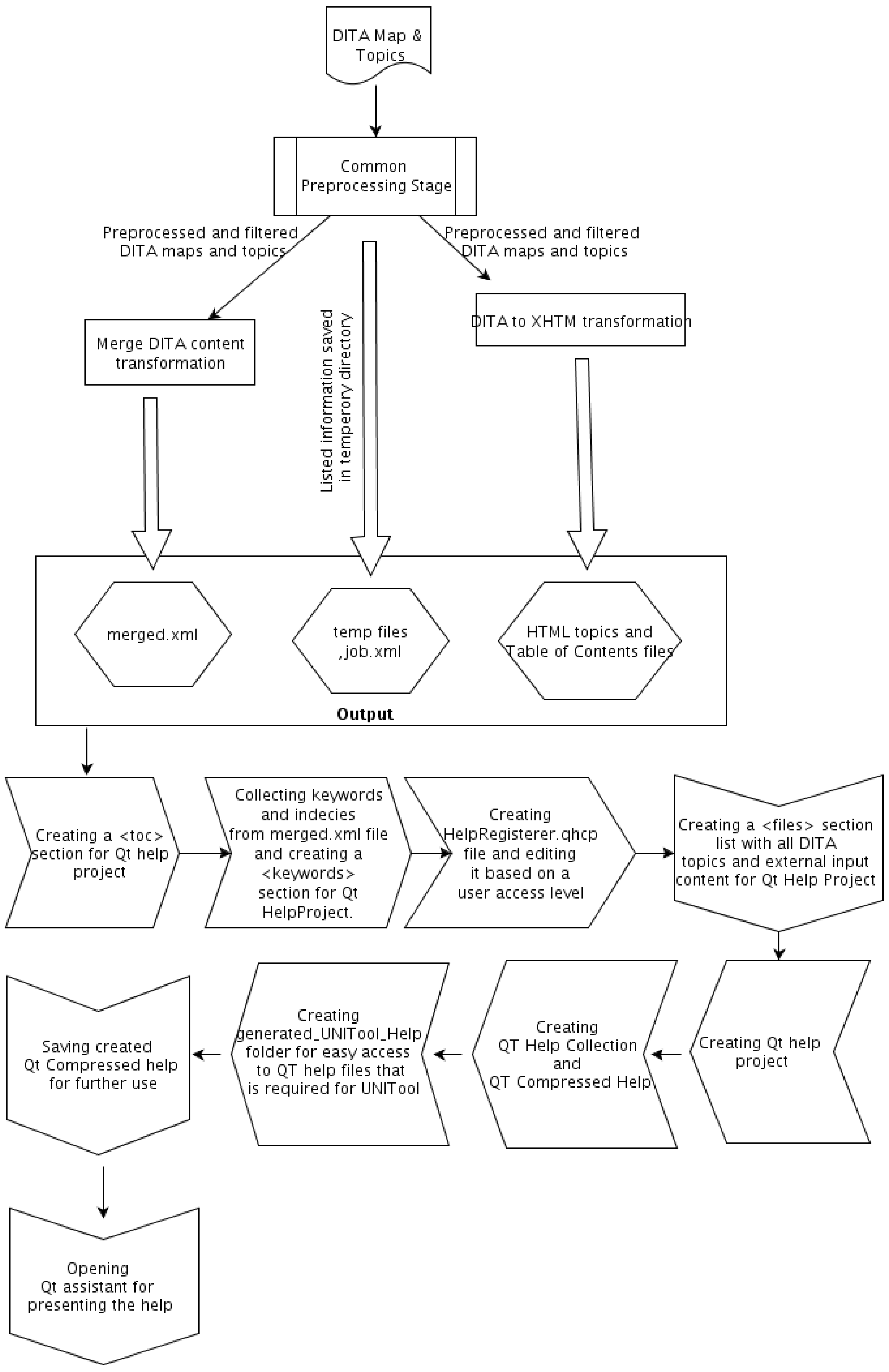

DITA Map & Topics
Common Preprocessing Stage
Preprocessed and filtered DITA maps and topics
Preprocessed and filtered DITA maps and topics
Listed information saved in temperory directory
Merge DITA content transformation
DITA to XHTM transformation
merged.xml
temp files ,job.xml
HTML topics and Table of Contents files
Output
Creating a <toc> section for Qt help project
Collecting keywords and indecies from merged.xml file and creating a <keywords> section for Qt HelpProject.
Creating HelpRegisterer.qhcp file and editing it based on a user access level
Creating a <files> section list with all DITA topics and external input content for Qt Help Project
Creating Qt help project
Creating QT Help Collection and QT Compressed Help
Creating generated_UNITool_Help folder for easy access to QT help files that is required for UNITool
Saving created Qt Compressed help for further use
Opening Qt assistant for presenting the help

Figure 12. The visual diagram of the transformation pipeline.

For integrating the Ant code to the main DITA-OT build process, the Ant script with transformation process definition shall contain a target based on the name of the transformation type. In current case the name of the target is dita2UNITool_Help.

Dita2UNITool_Help target does not contain any commands inside, but it specifies dependencies for the rest of the script. The target shall depend on each target defined in the file, additionally, it depends on dita2merge value. Dita2merge corresponds to the DITA-OT target for running "Merge DITA content" transformation.

Before the DITA-OT pre-processing stage, some of more commands were developed for troubleshooting purposes. The first command deletes the output directory if one exists, then it checks the directory for a DITAVAL file, and the input for a user access level. The tested values can be checked in the output console.

7.2.1 Dita2XHTML customization

Based on the help content requirements, a custom parameter was added to the plugin. User.level gets a value from a user input before the transformation is started. In Oxygen XML Author this function is implemented by a dialogue box. When running "DITA to UNITool Help" from command line, the user shall specify -Duser.level="<user level>" parameter. The user level can be one of the following: developer, expert, operator, or viewer. The scenario of processing user level variables is written in the ditaval files which will be described later.

Code Snipped 6. presented below defines the parameters for XHTML transformation. ${dita.dir} corresponds to the location of DITA-OT in a system, ${user.level} value is received from a user input.

Table 10. describes the parameters used for the customization of the "DITA to XHTML" transformation scenario.

Table 10. XHTML transformation parameters description.

Parameter name.	**Description.**
args.css	The custom style sheet files are located in DITA-OT\Plugins\UNITool_Help\css. For the correct representation of the help content in the UNITool help frame-work addedstyles.css must be applied to dita2XHTML transformation.
args.copycss	Copy the custom CSS files to the output directory.
args.csspath	The CSS files must be copied to the output directory: out/UNITool_Help/Help_source/Resources
args.filter	Following the UNITool help content requirements, the help source must be fil-tered based on the user access level. The filtering scenario is defined in the DITAVAL files that are located in DITA-OT\Plugins\UNITool_Help\filters.
args.xhtml.toc.xsl	Parameter for customizing ToC file. XHTML transformation creates in-dex.html for displaying the table of contents, the UNITool_Help transfor-mation override the XSLT scenario for index.html so later it can be used as the <toc> section in Qt Help Project file.
processing-mode	Strict processing mode for the transformation is set for fatal failure of the transformation on any error, this mode ensures an additional check of input content for users.
dita.dir	Dita.dir specifies the directory of the customized DITA-OT.

```
<target name="dita2UNITool_Help.init"
    depends="preprocessing"
    description="run the dita2XHTML transformation with custom parameters">

    <antcall target="dita2xhtml">
      <!-- Custom .css file used to style output -->
      <param name="args.css" value="${dita.dir}\plugins\UNITool_Help\css\addedstyles.css"/>
      <!-- Copy the custom .css file to the output directory -->
      <param name="args.copycss" value="yes"/>
      <!-- Location of the copied .css file relative to the output -->
      <param name="args.csspath" value="Resources"/>
      <!-- Location of the custom user access level .ditaval file -->
      <param name="args.filter" value="${plugin.dir}\filters\${user.level}.ditaval"/>
      <!-- Custom toc scenario for index.html -->
      <param name="args.xhtml.toc.xsl" value="${plugin.dir}\xsl\customXHTML\customToc.xsl"/>
      <!-- Processing-mode shall be set to "strict" for transformation -->
      <param name="processing-mode" value="strict"/>
      <!-- DITA OT dir -->
```

```
        <param name="dita.dir" value="C:\WADE\DITA-OT"></param>
      </antcall>
    </target>
```

Code Snippet 6. Source code of dita2UNITool_Help.init target.

7.2.2 DITA to Qt Help transformation

After the built-in DITA-OT transformations were completed, the next phase of the “DITA to UNITool_Help” scenario is the modification of the generated output files from “DITA to XHTML” and “Merge DITA content” transformations to appropriate input content for the Qt Help processor.

The first step is editing the index.html, the file that contains table of contents information, to the form that can be used in the Qt Help Project file. For that purpose, Ant filterchain commands were used. They replace certain words in index.html with corresponding Qt Help Project elements and save the result as toc.xml document. The following Code Snipped 7. contains the script for the actions described above. ${output.dir} is a variable for the directory where the output file will be saved, by default it is located within the same directory as the input master DITA map, under “out” folder.

```
<target name="create_toc"
      depends="dita2UNITool_Help.init"
      description="transform index.html to toc.xml; toc.xml has a toc for qt help project">

      <copy file="${output.dir}/index.html" tofile="${plugin.dir}\xsl\help_files\toc.xml">
         <filterchain>
            <linecontainsregexp>
               <regexp pattern="/QhpSection|QhpSection|section_title| ref|body| /body"/>
            </linecontainsregexp>
            <tokenfilter>
               <replacestring from="/QhpSection" to="/section"/>
               <replacestring from="QhpSection" to="section"/>
               <replacestring from="section_title" to="title"/>
               <replacestring from="body" to="toc"/>
            </tokenfilter>
         </filterchain>
      </copy>

   </target>
```

Code Snippet 7. Source code of “create_toc” target.

The second step is creating the <keywords> section for the Qt Help Project file. This section contains words or phrases that will be used as the index keywords in the output. In addition, due to UNITool help framework requirements, during this step all the output files are copied under Help_source folder. By applying the XSLT scenario findIndex.xsl to the merged.xml file, the index_list.xml document is created in the temporary directory ${plugin.dir}\xsl\help_files.

```
<target name="index"
    depends="create_toc"
    description="collect keywords and indices from merged file">

    <move todir="${output.dir}\Help_source">
      <fileset dir="${output.dir}"/>
    </move>
    <xslt in="${output.dir}\Help_source\${dita.map.filename.root}.xml"
      out="${plugin.dir}\xsl\help_files\index_list.xml"
      style="${plugin.dir}\xsl\findIndex.xsl"/>

  </target>
```

Code Snippet 8. Source code for the “index” template.

For the correct representation of the source in Qt help browser, all the files used as the input shall be listed in the <files> section. The .job.xml is one of the temporal files created during the XHTML transformation, it contains the list of DITA topics and images that were used as the input. By applying the files.xsl transformation scenario, the requested content is generated and saved as a file_list.xml. Code Snippet 9. contains the script for the actions described above.

```
<target name="get_file_list"
    depends="create_qhcp"
    description="collects all the files used in source dita documentation;
    .job.xml file in temp dir is used as a source;
    creates the 'files' section in .qhp">

    <xslt in="${dita.temp.dir}\.job.xml"
      out="${plugin.dir}\xsl\help_files\file_list.xml"
      style="${plugin.dir}\xsl\files.xsl"/>
```

```
</target>
```

Code Snippet 9. The source code for the “get_file_list” template.

There are separate targets for creating valid Qt Help Project and Qt Help Collection Project files. Qt Help Project is created by applying the qhp_creator.xsl transformation scenario. The Qhp_creator.xsl collects a title value from the merged.xml file and imports previously created sections, <toc>, <keywords>, and <files>, with collected information. The Qt Help Collection Project template is updated depending on the user level and the name of the input DITA map file. For editing the source of the Qt Help Collection Project file, Ant filterchain command is used.

The Qt Compressed Help and the Qt Help collection binariy files are generated with qcollectiongenerator. The generation is done by calling the Qt command qcollectiongenerator HelpRegisterer.qhcp -o QTHelpCollection.qhc in the Windows command line tool. Code Snippet 10. contains the exec command, which executes the command line tool, refers it to the directory where the Qt Help Project and the Qt Help Collection Project files were saved, and set the arg value for the command.

```
<target name="help_packages_creation"
   depends="qhp_creator"
   description="calls qt cmd and creates QT Help Collection .qhc and QT Compressed Help .qch">
   <exec executable="cmd"
      dir="${output.dir}"
      failonerror="true">
      <arg line="/C"/>
      <arg line="qcollectiongenerator HelpRegisterer.qhcp -o QTHelpCollection.qhc"/>
   </exec>
</target>
```

Code Snippet 10. Source code for “help_packages_creation” target.

The final steps of the transformation include copying the generated Qt files to a different parallel folder for easier access for the users. Additionally, for reviewing the generated content, Ant calls to open Qt Assistant where the information developers can see the documentation in the help browser similar to the UNITool help browser.

7.3 UNITool_Help plug-in. XSLT.

The information below describes the behavior and the logic of the XSLT scenario files referenced in build_unitool_help.xml. XSLT in DITA-OT is used for applying processing rules to DITA topics.

The XSLT modules in DITA-OT use shell files. For instance, dita2xhtml.xsl controls the XHTML processing. First, it imports common rules that can be applied to all general topics. Later, for applying processing specializations, additional XSLT overrides general scenario. After standard specializations are overridden, using plug-ins, the developers are able to add more processing rules for local styles or for additional specializations.

7.3.1 Table of contents customization.

For customizing the table of contents for the XHTML transformation, I chose to customize content of the shell file map2xhtml-cover.xsl by adding <include> command that refers to the qhp_toc.xsl, the file, which modifies the sequence responsible for creating the table of contents. For the UNITool_Help transformation the map2xhtml-cover.xsl file was copied and renamed as customToc.xsl. In the build_unitool_help.xml this file is referenced as a custom XSLT file for overwriting the built-in toc scenario.

Code Snippet 11. represents an example of the original content of the body container in index.html generated with the XHTML transformation.

```
<body><h1 class="title topictitle1">Test</h1><div>
    <ul class="map">
      <li class="topicref"><a href="topics/welcome.html">Welcome Page</a><ul>
          <li class="topicref"><a href="topics/info.html">Useful Information</a></li>
    </ul>
  </div></body>
```

Code Snippet 11. <body> section of the original index.html file.

Only the body part of the index.html is needed for creating the content for the <toc> container in Qt Help Project. The structure of the container is shown in Code Snippet 12.

```
<toc>
       <section ref="Help_source/topics/welcome.html" title="Welcome Page"/>
           <section ref="Help_source/topics/info.html" title="Useful Information"/>
       <section/>
 </toc>
```

Code Snippet 12. <toc> section of the Qt Help Project file.

The qhp_toc.xsl file is the customized map2htmlImpl.xsl file found in DITA-OT/plugins/org.dita.xhtml/xsl/map2htmtoc directory and copied to UNITool_Help plug-in. The original XSLT file contains the processing instructions for creating the <body> container of the index.html document. Index.html body contains the source for the table of contents. The data is formed using an unordered list, the bullets of the list contain links to the generated HTML pages. And the structure of the list repeats the tree structure of the content defined in the input DITA map.

The qhp_toc.xsl transformation replaces <li> elements with the <QhpSection> elements. The name of the element could not have been defined as <section> at this stage due to internal DITA-OT processing of the HTML elements. The <QhpSection> does not duplicate any HTML elements, therefore is not affected by the transformation and remains the same in the created output. Additionally, customized scenario excludes <a> elements and replaces the href attribute with ref. Help_source/ string is needed for covering the UNITool help framework requirement for recognizing the help source. Moreover, some additional commands were removed as not needed for the transformation.

7.3.2 Qt Help Project compilation.

For creating Qt Help Project content, several XSLT scenarios were used.

qhp_creator.xsl

The qhp_creator.xsl contains the core structure of Qt Help Project and imports toc.xml, index.list.xml and file_list.xml files that compile information for the <filterSection> container. Input file for the transformation is the merged.xml, output - the Qt Help Project file.

The values for the namespace, the customFilter and the filterAttribute elements for Qt Help Project are collected from the merged.xml as a title of a master input DITA map. The original title value needs to be modified because of the namespace value limitations. The namespace shall not contain any spaces or quote signs in a string, for that reason XSLT function replace was applied.

```
<xsl:variable name = "title" select="replace(/dita-merge/map/title/text(), '[^a-zA-Z0-9]', '_')"/>
```

Code Snippet 13. Replace function is applied on “title” value.

files.xsl

The input file for the files.xsl transformation is .job.xml, output file is file_list.xml. The .job.xml is a file created by the “DITA to XHTML” transformation and it is stored under a temporary directory. The files.xsl extracts file references from .job.xml and converts them to the format that is acceptable by Qt Help Project in the <files> section.

The .job.xml is created by DITA-OT during the pre-processing stage; it contains a list of all files that were referred in the input DITA map. Internal DITA and DITA map files are referenced with extensions .dita and .ditamap, external files are referenced with the extensions depending on the file format. The <files> section in Qt Help Project shall contain references to HTML data, therefore, the data collected from the .job.xml had to be modified. “DITA to XHTML” transformation creates a separate HTML page for every DITA topic while saving the name of the original file. The files.xsl replaces .dita extensions with .html, and adds a Help_source/ string to the beginning of the path.

```
<xsl:template match="//files/file">
```

```
        <xsl:if test="not(preceding::file[@uri = current()/@uri])">

          <xsl:if test="not(contains(@uri,'ditamap'))">
            <file>
              <!-- All helpsource must be under Help_source folder-->
              <xsl:text>Help_source/</xsl:text>
              <xsl:choose>
                <!-- Replacing .dita extension to .html -->
                <xsl:when test="@uri[substring(., string-length()-4)='.dita']">
                  <xsl:variable name="num" select="string-length(@uri)-4"/>
                  <xsl:value-of select="concat(substring(@uri,1, $num),'html')"/>
                </xsl:when>
                <xsl:otherwise>
                  <xsl:value-of select="@uri"/>
                </xsl:otherwise>
              </xsl:choose>
            </file>
          </xsl:if>
        </xsl:if>
      </xsl:template>
```

Code Snippet 14. XSLT template for catching and modifying source for <files> section.

findIndex.xsl

The output of the findIndex.xsl transformation is the index_list.xml document. The index_list.xml contains the content for the <keywords> section in Qt Help Project. The input file for the findIndex.xsl transformation is the merged.xml.

An index is a word or a short phrase defined by a documentation writer using DITA elements <keyword> or <indexterm>. For realizing the context sensitivity feature for the UNITool help framework, the <keywords> section in Qt Help Project shall contain references to every HTML page that is included into the help content with specific, unique IDs.

The findIndex.xsl transformation creates two types of the <keyword> elements that will be included in the <keywords> section. The first type is used for the basic DITA elements. The template for the action is shown in Code Snippet 17. The path reference to the topic where the <keyword> or the <indexterm> element was found in the merged.xml. Extracting wanted part out of an @xtrf attribute value was a problematic task due to the complexity of the DITA-OT processes for creating a temporary directory for locating input DITA files. The path to the file in the merged.xml is shown in Code Snippet 15, and the requested format of the link for Qt Help Project is shown in Code Snippet 16.

```
<indexterm class="- topic/indexterm "
        xtrf="file:/C:/Users/ADO021/AppData/Local/Temp/OxygenXMLTemp/ditaTemporaryOutputDir-
1508754533387/doc/common/app_desc/topics/purpose_of_app_description.dita"
        xtrc="indexterm:1;6:24">index</indexterm>
```

Code Snippet 15. The <indexterm> element found in the merged.xml file.

```
<keyword name="index"
            ref="Help_source/doc/common/app_desc/topics/purpose_of_app_description.html"/>
```

Code Snippet 16. The <keyword> element in Qt Help Project.

The issue was solved by applying an XSLT tokenize function shown in Code Snippet 22. The full.dita.map.dir variable catches a temporary directory path to the root DITA map, which in the current case will be "file:/C:/Users/ADO021/AppData/Local/Temp/OxygenXMLTemp/ditaTemporaryOutputDir-1508754533387". The index.dir variable selects the part of the @xtrf link after full.dita.map.dir.

```
<!-- Collects keywords and indexterms from prolog and any keywords found in dita source -->
<xsl:template match="//keywords/indexterm | //keyword">
  <xsl:variable name="full.ditamap.dir"
    select="ancestor::node()//map/substring-before(@xtrf, tokenize(@xtrf, '/')[last()])"/>
  <xsl:variable name="index.dir"
    select="concat('Help_source/', (substring-after(@xtrf, $full.ditamap.dir)))"/>

  <keyword>
    <xsl:attribute name="name">
      <xsl:value-of select="./normalize-space()"/>
    </xsl:attribute>
    <xsl:if test="not(preceding::xtrf[text() = current()/text()])">
      <xsl:attribute name="ref">
        <xsl:value-of select="replace($index.dir,'dita','html')"/>
      </xsl:attribute>
    </xsl:if>
  </keyword>
</xsl:template>
```

Code Snippet 17. Template for creating a <keyword> element.

The second type of the <keyword> element is created for activating the context sensitivity feature. For that reason, every <keyword> element shall additionally have a unique ID. The ID must be the same as a specified @id attribute in every input DITA topic.

7.3.3 Additional resources

Filtering

Filtering input DITA content is based on the audience attribute and controlled in the ditaval files. Every audience behaviour is controlled by a separate file. For instance, filtering instruction for the "operator" user access level for UNITool is controlled by operator.ditaval file. "Operator" has the access to the information available for "operator" and "viewer" users, and does not have the access to information available for "developer" and "expert". Code Snippet 18. contains the source code for the operator.ditaval.

```
<?xml version="1.0" encoding="UTF-8"?>
<val>
  <prop att='audience' val='developer' action='exclude' />
  <prop att='audience' val='expert' action='exclude' />
  <prop att='audience' val='operator' action='include' />
  <prop att='audience' val='viewer' action='include' />
</val>
```

Code Snippet 18. The source code of the operator.ditaval

The root element of the ditaval file is the <val>. A <prop> element with an @att attribute and a @value attribute sets an @action for that value within that attribute.

8 TESTS AND ANALYSIS

Based on the project specifications, a functional testing was chosen as the most appropriate technique for testing the features of the developed transformation. Test cases for the “DITA to UNITool Help” transformation testing were created based on the requirements specified for the project.

8.1 Testing fundamentals

Functional testing for the following modules are in the scope:

- Authoring
- Publication usability
- Build automation
- Technical specifications of the interface to UNITool help framework system

Testing environments are:

- Platform:
    - Windows 7, Windows 10
- DITA-OT 2.2.4
- Oxygen XML Author 19
- Qt Creator

8.2 Test cases

Every requirement defined for the software is a test situation, every test situation occurs in one test case. The test case is connected to a corresponding requirement by ID specified in chapter 5.1 of the current document.

The test cases for the “DITA to UNITool Help” transformation were created as presented in Table 11.

Table 11. The test case table.

Linked requirement by ID: R1	**Status:** Passed
Test case title: Verify: Qt Assistant for viewing the transformed UNITool help publication shall be opened automatically after the transformation is completed	
Test Case Description: Check that a transformation scenario for UNITool help will open the generated publication after the transformation has been completed	
Success Scenario: Qt Assistant is opened and generated documentation is available for viewing. In master DITA map folder location under out/generated_UNITool_Help/'user_level' QCH_filename_'user_level'.qch, QTHelpCollection.qhc files, and saved_help_collection folder are generated.	
Linked requirement by ID: R2	**Status:** Passed
Test case title: Verify: All UNITool help transformation related resources shall be moved under “UNITool help” DITA-OT plugin.	
Test Case Description: Check that all “UNITool help” transformation related resources are moved under “UNITool help” DITA-OT plug-in, including build and post-processing scripts, xslt-stylesheets, CSS and image files.	
Success Scenario: No files are found outside of the “UNITool_Help” plugin folder.	
Linked requirement by ID: R3	**Status:** Failed
Test case title: Verify: Index keyword generation for UNITool help publication shall follow the general DITA conventions for creating an index.	
Test Case Description: Check that index keywords in Qt Assistant are generated from general DITA index term element. General DITA conventions for creating an index means that indices are created from the <indexterm> elements. Only terms that were defined in the input DITA content as the <indexterm> or as the <keyword> elements are shown in the index tab in a help browser. The <keyword> element shall be included due to client request.	

Success Scenario: Qt Assistant shows the list of indices. Index list contains only terms that were defined in the input DITA content as the <indexterm> or the <keyword> elements

Linked requirement by ID: R4 **Status:** Failed

Test case title: Verify: UNITool help publication DITA processing shall support content filtering by a UNITool user profile(operator/service)

Test Case Description: Check if "DITA to UNITool Help" transformation supports filtering by a UNITool user profile: service, developer.

Success Scenario: Writers are able to specify the attribute with the values "service" or "developer" for creating content filtering instructions based on the UNITool user profile.

Linked requirement by ID: R5 **Status:** Passed

Test case title: Verify: Default processing mode of UNITool help transformation scenario shall be "strict"

Test Case Description: Check if "UNITool help" transformation scenario is 'strict'. Processing-mode strict refers to how the DITA-OT handles errors and error recovery. Allowed values for processing-mode:

"strict" - When an error is encountered, the DITA-OT stops processing

"lax" (default) - When an error is encountered, the DITA-OT attempts to recover from it

"skip" - When an error is encountered, the DITA-OT continues processing but does not attempt error recovery

Success Scenario: The "DITA to UNITool Help" transformation fails if the input documents contain a validation error.

Linked requirement by ID: R6 **Status:** Passed

Test case title: Verify: UNITool help transformation shall support content filtering by UNITool user access level (viewer, operator, expert, developer)

Test Case Description: Check if "DITA to UNITool Help" transformation supports filtering by a UNITool user access level: developer, expert, operator, and viewer.

Success Scenario: Writers are able to specify the attribute with the values “developer”, “expert”, “operator”, or “viewer” for creating content filtering instructions based on the UNITool access level.

Linked requirement by ID: R7 **Status:** Passed

Test case title: Verify: “UNITool help” transformation shall be possible to run using DITA-OT command line tool.

Test Case Description: Check that the “UNITool help” transformation can be run using DITA-OT command line tool.

Success Scenario: “UNITool help” transformation can be completed using DITA-OT command line tool.

Linked requirement by ID: R8 **Status:** Passed

Test case title: Verify: The help publication contents shall be designed based on UNITool user access levels (viewer, operator, expert, developer)

Test Case Description: Check if Qt Help information developers are able to specify audience attribute for UNITool user profile: developer, expert, operator, and viewer.

Success Scenario: The options for an @audience attribute are presented as “developer”, “expert”, “operator”, and “user”, @audience attribute is the valid element in the DITA document

Linked requirement by ID: R9 **Status:** Passed

Test case title: Verify: The UNITool help publication shall be passed to UNITool help browser as Qt Compressed Help (.qch) and Qt Help Collection (.qhc) that are generated with WADE

Test Case Description: QT Help Packages are used for representation help content for UNITool via Qt Help Browser. For successful representation of the help content DITA to UNItool Help transformation shall deliver Qt Compressed Help (.qch file) and Qt Help collection (.qhc file) generated based on the input DITA content.

Success Scenario: In master ditamap folder location under out/generated_UNITool_Help/'user_level' QCH_filename_'user_level'.qch, QTHelpCollection.qhc files, and saved_help_collection folder are generated.

Linked requirement by ID: R10 **Status:** Passed

Test case title: Verify: Each help publication topic shall have some unique identifier that will be used as Help ID transformed in Qt Compressed Help (.qch) package. Help ID is used in UNITool help framework for activating context sensitivity feature.

Test Case Description: Check Qt Help Project source code. Some <keyword> elements in the <keywords> container has the same unique identifier and the titles as in a source topic.

Success Scenario: Qt Help Project file has some source, <keywords> section can be located. Every value for id attribute in <keyword> element is the same as the `id` attribute in DITA topics from input source.

Linked requirement by ID: R11 **Status:** Passed

Test case title: Verify: UNITool help transformation fails if the DITA source does not meet the DITA-OT's limitations for transforming HTML-based outputs.

Test Case Description: The DITA source for UNITool help publication shall meet the UNITool processing limitations for creating Qt Help Packages
- no <topichead> elements are allowed DITA map.

Success Scenario: If the <topichead> element is present in the input DITA map, ttransformation fails with the error: "An empty sequence is not allowed as the value of parameter $filename."

Linked requirement by ID: R12 **Status:** Passed

Test case title: Verify: Authors who wish to transform UNITool help shall install Qt resources in addition to WADE.

Test Case Description: Qt Creator shall be installed for generating help content for UNITool
Success Scenario: The information about the requirement of additional software downloading can be found in the installation guide.

Linked requirement by ID: R13 **Status:** Passed

Test case title: Verify: UNITool help DITA-OT plugin shall be based on DITA v1.3
Test Case Description: Check that DITA-OT supports DITA v1.3
Success Scenario: The elements introduced in DITA version 1.3 are valid.

8.3 Analysis

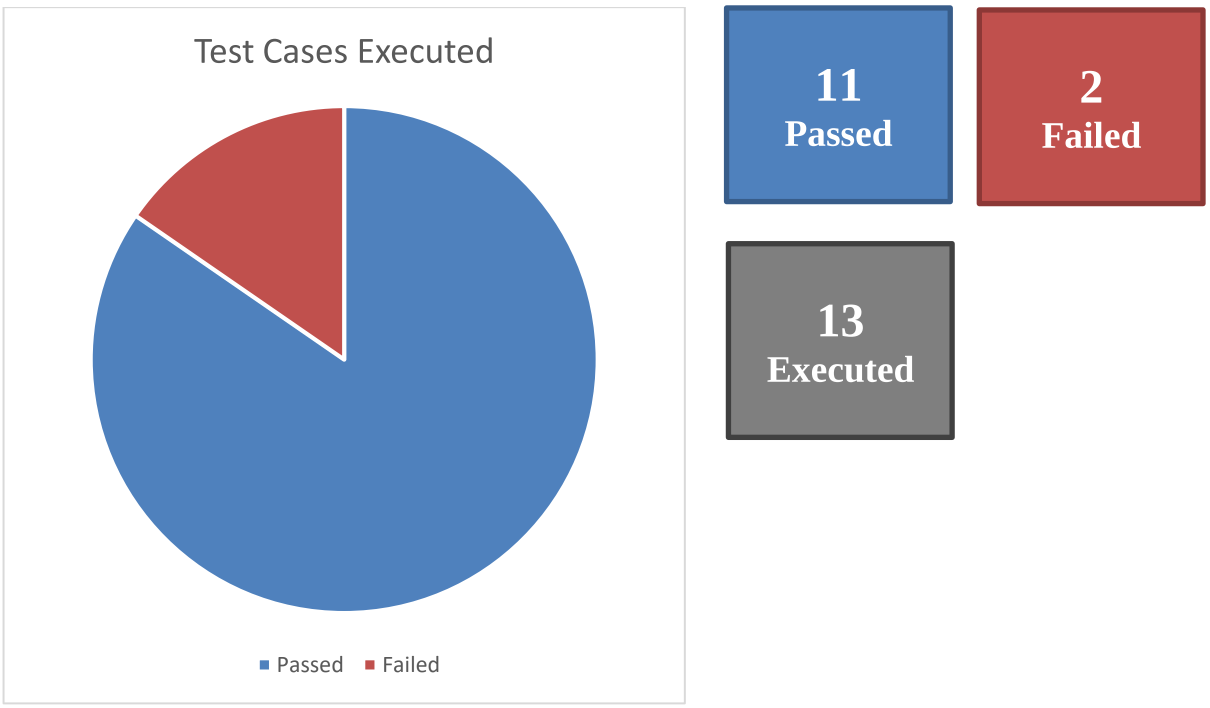


Figure 13. Visual result of the test cases execution.

During the functional testing thirteen test cases were executed. Eleven of the test cases passed successfully, two test cases have failed.

The following Table 13. contains the analysis of the test cases that failed during the testing phase.

Table 12. Failed test cases analysis.

Linked requirement by ID: R3	**Priority:** Highly Recommended
Failed test case title: "Verify: Index keyword generation for UNITool help publication shall follow the general DITA conventions for creating an index."	

Test case verdict: the expected result was to find that Index list in Qt help browser contains only terms that were defined in the input DITA content as indexterm elements; actual result – the list contains the titles of the input DITA topics.

Analysis: Analysis of the failure brought me to a conclusion that the failure is caused by UNITool help framework configuration, and it cannot be fixed in the scope of the current thesis

Linked requirement by ID: R4 **Priority:** Highly Recommended

Failed test case title: “Verify: UNITool help publication DITA processing shall support content filtering by UNITool user profile”

Test case verdict: The expected result was to find a functional support of filtering the source for UNITool help based on a user profile: developer/service.

Analysis: The test case is failed due to absence of final solution of the filtering scenario for the given values from the customer.

The result of the testing can be considered as successful.

9 CONCLUSION AND DISCUSSION

The main target of this thesis was to develop a DITA-OT transformation scenario that generates Qt help files that can be used as a source for displaying help content in UNITool application from the input DITA content.

The development of the project involved deep learning of XML, XSLT, and Apache Ant programming languages, investigating and studying of the DITA standard philosophy, the DITA-OT structure and customization techniques. During the empirical study of this thesis I faced several difficulties when exploring DITA-OT internal processes, as some of the steps in the preprocessing and later stages of generating outputs are implemented with Java, and I could not find a way to get there or separate the output that would only contain the source that I need.

However, as the conclusion, I would like to state that DITA Open Toolkit has a great customization and developing potential. It has a good structure and documented customization techniques with a wide range of possibilities for companies to create new transformation scenarios that will extend the profitability and the usability of the documentation written in DITA.

9.1 Further development

Further development of the current project includes:

- process optimization of generating the Qt help packages for UNITool Help framework in a way, that the information developer is able to generate the help source for all user access level during one run;
- solving the issue with resolving the file references that are located outside of the root map directory;
- finding a solution for the index generating issue, caused by the UNITool help framework implementation.

10 TERMINOLOGY

10.1 Index

ANT	Apache Ant, ANT stands for Another Neat Tool, is a Java library and a command line tool
C++	A compiled language.
DITA	Darwin Information Typing Architecture, is an XML data model for authoring and publishing.
DITA-OT	DITA Open Toolkit, is a publishing engine for content authored in DITA.
Help ID	Unique identifier found in every page of documentation content, used as a reference to the corresponding help information from the user interface of UNITool.
HTML	HyperText Markup Language. The programming language in which Internet pages are formulated.
Qt	A cross-platform application framework written on C++.
Qt Help Framework	A Qt-based help system which includes tools for viewing and generating Qt help files, and the methods for integrating help into Qt applications.
UNITool	Maintenance tool for downloading, tuning, monitoring, testing and troubleshooting module software in UNIC.
XML	Extensible Markup Language. XML documents describe a class of data objects, and, to a limited extent, the behavior of computer programs that process the documents.

XSLT	Extensible Stylesheet Language: Transformation. The programming language used for transforming one XML document to another.

11 REFERENCES

/1/ JoAnn T. Hackos. 2011. Introduction to DITA- Second Edition: A User Guide to the Darwin Information Typing Architecture Inc (2nd Second Edition)

/2/ Laura Bellamy, Michelle Carey, Jenifer Schlotfeldt. 2012. DITA Best Practices. A Roadmap for Writing, Editing, and Architecting in DITA.

/3/ Companies Using DITA – Last access 11.10.2017

http://www.ditawriter.com/companies-using-dita/

/4/ Processing structure – Last access 11.10.2017

http://www.dita-ot.org/2.5/dev_ref/processing-structure.html

/5/ Extensible Markup Language (XML) 1.0 – Last access 11.10.2017

https://www.xml.com/axml/axml.html

/6/ Michael kay.2000. XSLT Programmer's Reference.

/7/ Qt Documentation – Last access 11.10.2017

http://doc.qt.io/qt-5/qtassistant-index.html

/8/ Resolve keyref (keyref) – Last access 11.10.2017

http://www.dita-ot.org/2.2/dev_ref/preprocess-keyref.html

/9/ DITAVAL elements – Last access 11.10.2017

https://docs.oasis-open.org/dita/v1.2/os/spec/common/about-ditaval.html