



Gemalto Customer Documentation 2.0

MADCAP FLARE TEMPLATE USER GUIDE



Document Information

Product Version	2.0
Document Number	<PartNumber>
Release Date	18 April 2019

Revision History

Revision	Date	Reason
Rev. A	18 April 2019	Initial release

Trademarks, Copyrights, and Third-Party Software

Copyright 2019 Gemalto. All rights reserved. Gemalto and the Gemalto logo are trademarks and service marks of Gemalto and/or its subsidiaries and are registered in certain countries. All other trademarks and service marks, whether registered or not in specific countries, are the property of their respective owners.

Disclaimer

All information herein is either public information or is the property of and owned solely by Gemalto and/or its subsidiaries who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemalto's information.

This document can be used for informational, non-commercial, internal, and personal use only provided that:

- > The copyright notice, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- > This document shall not be posted on any publicly accessible network computer or broadcast in any media, and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemalto makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemalto reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemalto hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemalto be liable, whether in contract, tort or otherwise, for any indirect, special or

consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemalto does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemalto be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Gemalto products. Gemalto disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service, or loss of privacy.

All intellectual property is protected by copyright. All trademarks and product names used or referred to are the copyright of their respective owners. No part of this document may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, chemical, photocopy, recording or otherwise without the prior written permission of Gemalto.

CONTENTS

Preface: About the Gemalto Flare Template	7
Audience	7
Release Notes	7
Document Conventions	8
Command Syntax and Typeface Conventions	8
Notifications and Alerts	8
Related Documents	9
Support Contacts	9
Customer Support Portal	10
Telephone Support	10
Email Support	10
Chapter 1: Template Overview	11
The Importance of Templates	11
The Characteristics of Good Documentation	11
How This Template Provides the Tools Required to Create Good Documentation	12
About the Template	12
Branding	12
Features	12
Template Inputs	14
Template Outputs	14
Flare Workflow	15
Content	16
Tables of Contents	17
Targets	18
Outputs	19
Single-Sourcing	19
How This Template Uses Single Sourcing	19
Chapter 2: Creating Content	21
Authoring Best Practices	21
Creating a New Book	21
Structuring Your Content	22
Building the Book	23
Creating Chapters	23
Headings	24
Level-1 Headings	24
Level-2 Headings	25
Level-3 Headings	25
Level-4 Headings	25
Level 5 Headings	25
Generic Subclasses	25

Inserting Headings Into a Topic	26
Paragraphs	26
Paragraph Style Classes Used to Create Paragraphs in Text	26
Paragraph Style Classes Used to Create Lists	26
Paragraph Style Classes Used in Table Cells	26
Paragraph Style Classes Used for Examples and Code Samples	26
Generic Subclasses	27
Notes, Cautions, Warnings, and Tips	27
Notes	27
Cautions	28
Warnings	28
Tips	29
Inserting a single-paragraph notification using a paragraph style	29
Inserting a multi-paragraph notification using a div style	30
Examples and Command and Code Snippets	31
Using Preformatted Text	33
Using Syntax Highlighting for Code Samples	33
Lists	40
Ordered Lists	40
Unordered Lists	41
Tables	42
Paragraph Formats Supported in Table Cells	43
Using Tables in Body Text	44
Using Tables in Lists	46
Inserting Tables	46
Procedures	48
Creating a Procedure	48
Example Procedure	49
Figures and Graphics	51
Storing your Graphic Images	51
Graphic Formats	51
Sizing Your Graphics	51
Inserting Figures	52
Using Variables	52
The Default Variable Set	52
Cross-References and Hyperlinks	54
Cross-References	54
Hyperlinks	55
Creating an Index	55
Entering Index Items	55
Including an Index in a PDF Book	55
Creating a Glossary	56
Entering Glossary Terms and Definitions	56
Including the Glossary as an Appendix in a Single-Book Project	56
Including the Glossary as a Separate Book in a Multiple-Book Project	56
Creating the Home Page	57

Chapter 3: Importing Content	59
Topic-Based Authoring	59
Chunking Strategy	60
Chunking Methodology	60
Importing MS Word Documents	60
Preparing Your MS Word Documents for Import	61
Using the MS Word Import Wizard	62
Post-Import Cleanup and Configuration in Flare	63
Chapter 4: Generating Outputs	65
TOCs: Defining the Output Contents, Structure, and Layout	66
PDF TOCs	67
HTML TOCs	70
Configuring Your Output Targets	73
Creating the PDF Targets	74
Creating the HTML Target	75
Creating Batch Targets	78
Setting Up Your Project for Combined HTML and PDF Outputs	79
Building and Publishing Output Targets	80
Outputs that combine HTML and PDF	80
Manual Building and Publishing	81
Automated Building and Publishing	81
Publishing to a Web Server	83
Appendix A: List of Styles	84
Generic Classes	84
.ntoc	84
.pagebreak	84
Heading Styles	84
Paragraph Styles	86
Appendix B: Glossary of Terms	90
Index	94

PREFACE: About the Gemalto Flare Template

This document describes the Gemalto single-sourcing Madcap Flare customer documentation template and how to use this template to generate HTML and PDF from a single set of source files.

This document contains the following chapters:

- > ["Template Overview" on page 11](#)
- > ["Creating Content" on page 21](#)
- > ["Importing Content" on page 59](#)
- > ["Generating Outputs" on page 65](#)

This preface also includes the following information about this document:

- > ["Release Notes" below](#)
- > ["Audience" below](#)
- > ["Document Conventions" on the next page](#)
- > ["Related Documents" on page 9](#)
- > ["Support Contacts" on page 9](#)

For information regarding the document status and revision history, see ["Document Information" on page 2](#).

Audience

This document is intended for personnel responsible for maintaining your organization's security infrastructure. This includes Luna HSM users and security officers, key manager administrators, and network administrators.

All products manufactured and distributed by Gemalto are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

It is assumed that the users of this document are proficient with security concepts.

Release Notes

The release notes provide important information about this release that is not included in the customer documentation. It is strongly recommended that you read the release notes to fully understand the capabilities, limitations, and known issues for this release. You can view or download the latest version of the release notes for this release at the following location:

<insert_link_here>

Document Conventions

This section describes the conventions used in this document.

Command Syntax and Typeface Conventions

This document uses the following conventions for command syntax descriptions, and to highlight elements of the user interface.

Format	Convention
bold	<p>The bold attribute is used to indicate the following:</p> <ul style="list-style-type: none"> > Command-line commands and options that you enter verbatim (Type dir /p.) > Button names (Click Save As.) > Check box and radio button names (Select the Print Duplex check box.) > Dialog box titles (On the Protect Document dialog box, click Yes.) > Field names (User Name: Enter the name of the user.) > Menu names (On the File menu, click Save.) (Click Menu > Go To > Folders.) > User input (In the Date box, type April 1.)
<i>italics</i>	In type, the italic attribute is used for emphasis or to indicate a related document. (See the <i>Installation Guide</i> for more information.)
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[optional] [<optional>]	Represent optional keywords or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task.
{a b c} <a> <c>	Represent required alternate keywords or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.
[a b c] [<a> <c>]	Represent optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.

Notifications and Alerts

Notifications and alerts are used to highlight important information or alert you to the potential for data loss or personal injury.

Tips

Tips are used to highlight information that helps to complete a task more efficiently.



TIP This is some information that will allow you to complete your task more efficiently.

Notes

Notes are used to highlight important or helpful information.



NOTE Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss.



CAUTION! Exercise caution. Contains important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury.



****WARNING**** Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.

Related Documents

The following documents contain related or additional information:

- > cross-reference to the **About.htm** file for the related document
- > cross-reference to the **About.htm** file for the related document
- > cross-reference to the **About.htm** file for the related document



NOTE Use of this topic is optional. It is not required if any related documents are included in the same documentation package, or if there are no related documents.

Support Contacts

If you encounter a problem while installing, registering, or operating this product, please refer to the documentation before contacting support. If you cannot resolve the issue, contact your supplier or **Gemalto Customer Support**.

Gemalto Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Gemalto and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at <https://supportportal.gemalto.com>, is where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable database of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.



NOTE You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone Support

If you have an urgent problem, or cannot access the Customer Support Portal, you can contact Gemalto Customer Support by telephone at **+1 410-931-7520**. Additional local telephone support numbers are listed on the support portal.

Email Support

You can also contact technical support by email at technical.support@gemalto.com.

CHAPTER 1: Template Overview

This chapter describes the rules and best practices to use when authoring documentation. Adhering to these rules/best practices ensures documentation consistency. This chapter contains the following sections:

- > ["The Importance of Templates" below](#)
- > ["About the Template" on the next page](#)
- > ["Flare Workflow" on page 15](#)
- > ["Single-Sourcing" on page 19](#)

The Importance of Templates

Creating documentation is not difficult, if the only objective is to be able to say that the product includes documentation. However, when documentation is treated as an afterthought and an unimportant part of the product, the costs to the organization can be significant. Documentation that is inaccurate, incomplete, inconsistent, poorly formatted, and generally of poor quality can cost the organization in the following ways:

- > increased support costs. If the users cannot use the documentation to quickly and easily find the answers to their questions/problems they will call support, at much greater cost to the organization.
- > perception of the company as unprofessional. If the documentation for different products looks different, is structured differently, or varies in quality between different products, the user may question the professionalism of the organization, causing them to consider other vendors.

Rather than being a necessary cost of doing business, good documentation can reduce costs, increase business, and enhance customer satisfaction and loyalty - all things that can impact the bottom line.

The Characteristics of Good Documentation

Creating usable, useful and helpful documentation requires the right tools and processes, planning, and adherence to a set of rules that can serve as a means test, or checklist, for assessing the quality the documentation. Some of the characteristics that you should strive for when creating technical documentation are as follows:

Accurate	Correctly describes the software application. Mistakes in documentation confuse users and hinder the users' ability to use the system.
Complete	Covers all topics that users need to know to use the product effectively and efficiently.
Clear	Quickly read and easily comprehended by readers. Standard terminology.
Easily located	Easy to find (navigation, search, index, context-sensitive help)

Single-sourced	Information is maintained in one place only. It can be used in, or referenced from, multiple documents in multiple different formats
Task-oriented	Focused on tasks the user needs to perform. Every piece of information must support user tasks. Otherwise, it should be removed from the documentation.
Minimal	Contains only the information the user needs. Minimalist does not mean the documentation lacks necessary detail. It does mean that the documentation does not contain extraneous information.
Efficient	Tools and processes allow the writers to produce documentation efficiently

How This Template Provides the Tools Required to Create Good Documentation

This template supports single-sourcing, and uses a minimal set of styles to ensure documentation consistency and efficient authoring. The template workflow creates a package that includes all of the documentation in both HTML and PDF formats that can be deployed on product, to a web server, file server, or DVD/CD.

The documentation for this template includes instructions for configuring the tools and workflow to enable complete, accurate, well structured, task-oriented, minimalist documentation that helps your readers to successfully learn a concept, perform a task, or solve a problem.

About the Template

The template described in this document allows you to create multiple outputs from a single set of source documents using the MadCap Flare authoring platform. These outputs can be different formats (currently PDF and HTML outputs are supported), or contain content intended for a specific audience, for example to provide customer-specific documentation variants, or internal-only documentation variants.

The content is divided into a series of books. Each book is output as a single PDF file. The HTML output contains all of the books, with each book contained within a folder in the HTML table of contents.

In addition to supporting multiple outputs, this template automates the packaging of the outputs into a single user interface that provides access to all of the documentation in both HTML and PDF formats. All of the components used to create the outputs, including the user interface, are created directly within Flare, with no post-processing required.

Branding

The template is available in all of the Gemalto brand colors. The only difference is the use of the primary branding color in the skin, headings, cover, and notifications. This document uses the Enterprise (purple) branding.

Features

This template is designed to be as simple to use as is possible using Flare. Flare provides a multitude of features and possible workflows. Every attempt has been made to use the available features and workflows to stress simplicity and efficiency, as follows:

Stylesheets

This template uses a single stylesheet (Gemalto_Template.css) to define the look and feel of all of the outputs: Books, CRNs and Technical Notes in both HTML and PDF format.

Page Layouts

Page layouts are used to generate your print outputs. This template provides a set of standard page layout and a set of page layouts that watermark each page in the document with a Draft or Confidential watermark.

Both the standard and watermark page layouts use the following page layouts:

Chapters	Used for chapters, appendices, and contents. It has two page types: <ul style="list-style-type: none"> > First - for the h1 topic that begins the chapter > Normal - for all h2 topics in the chapter
Frontmatter	Used for the Cover and Document Information. It has two page types: <ul style="list-style-type: none"> > Title - for the document cover > Normal - for the document information page.
Index	Used for the Index, which uses a two-column format. It has two page types: <ul style="list-style-type: none"> > First - for the first page of the index. > Normal - for subsequent pages.
Notes	Used for Technical Notes. It has two page types: <ul style="list-style-type: none"> > First - for the first page of the note. > Normal - for subsequent pages.
RelNotes	Used for Release Notes. It has two page types: <ul style="list-style-type: none"> > First - for the first page of the release note. > Normal - for subsequent pages.

The headers and footers are automatically populated with variables, and require no manual configuration.

Single-Sourcing

You can create an integrated set of outputs that include PDF and HTML versions of your documents all from a single set of content files. A splash (Home) page provides links to all of the documents in all formats. The output is generated using a single command and can be published on-product, to DVD or CD, the web, or a file server without requiring any manual intervention.

Variables

A common set of variables is available for all documents. The variable is defined separately for each individual output target.

Glossaries

Support is provided for a glossary at the book level (as an appendix) or at the project level (as a separate book).

Drag and Drop Book Building

A default book includes default elements (chapters, topics, appendices, etc), that you can drag and drop to create new books.

HTML Skins

You can generate your HTML outputs using a side-navigation (HTML_FRAMES) or top-navigation (HTML_FRAMELESS) skin, as illustrated in ["Template Outputs" below](#).

Template Inputs

The intent of this template is to provide a set of tools that satisfy the requirements for creating user-friendly, technically accurate documentation as efficiently as possible. To satisfy these requirements, this template not only provides a consistent look and feel, but a structure and workflow that helps to ensure consistency in the way the documentation is authored, structured, and delivered.

Template Outputs

The output created using this template includes a Home page, or splash screen, that is displayed when the user launches the documentation application, as illustrated below for both the side-navigation and top-navigation skins. The Home page is an HTML file you maintain in the project and output as part of the HTML build. It provides links to all of the documents in the project.

Figure 1: The Home page using the side-navigation (HTML_FRAMES) skin

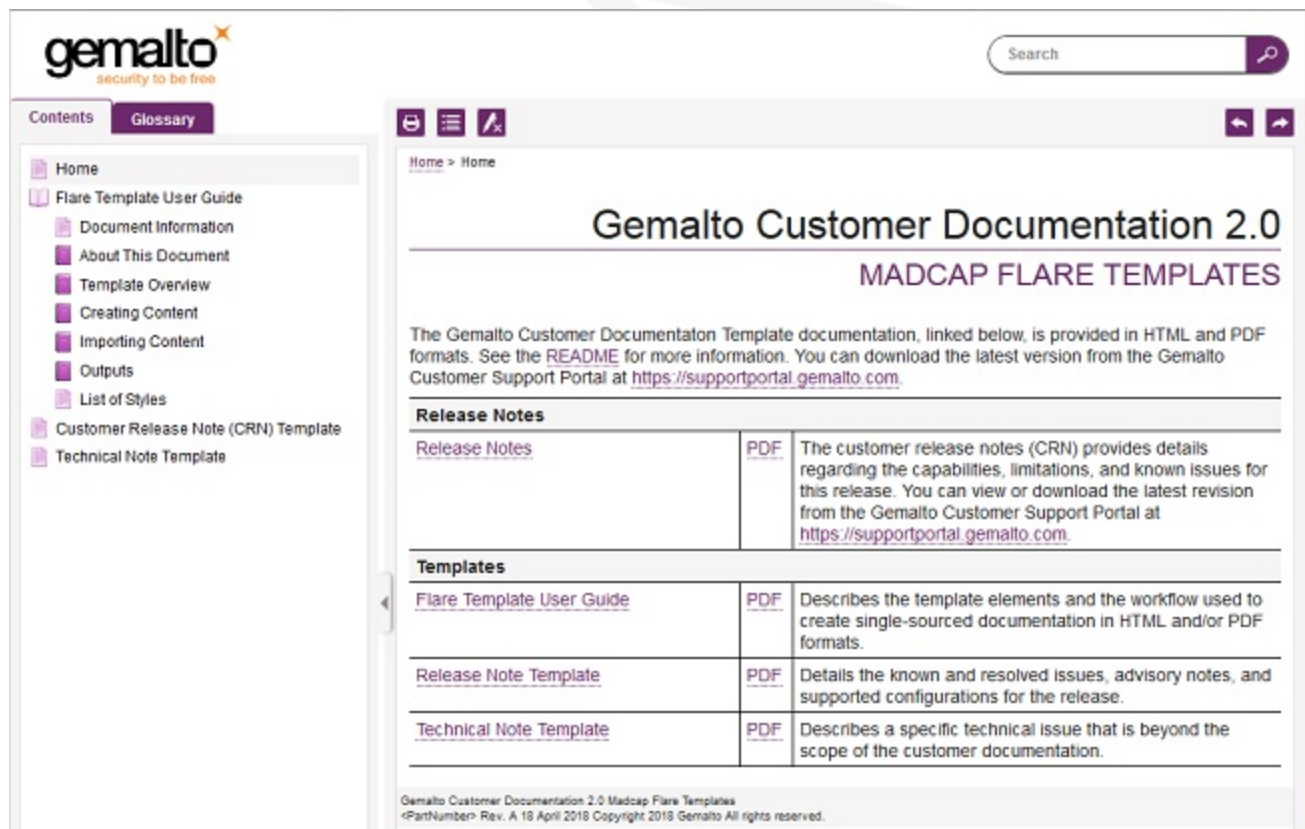
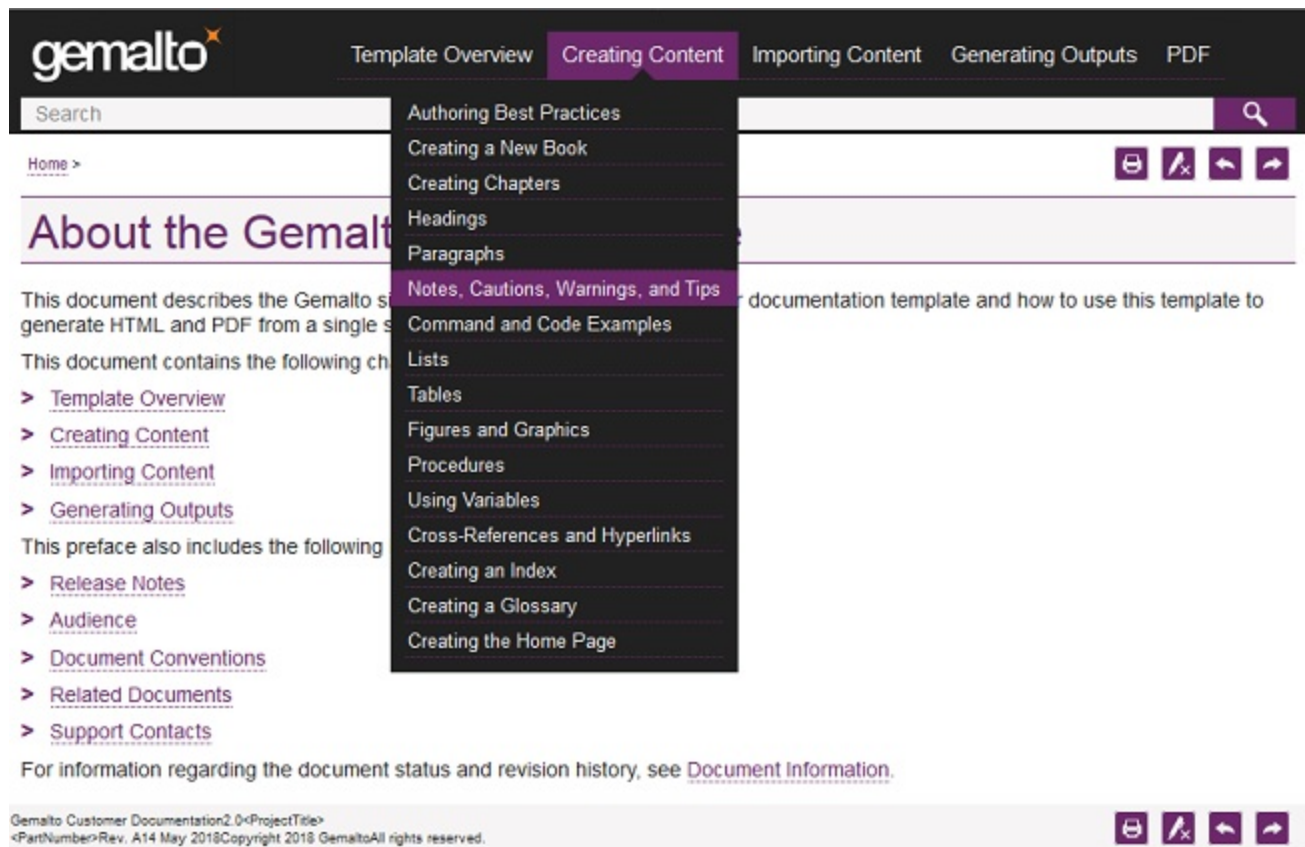


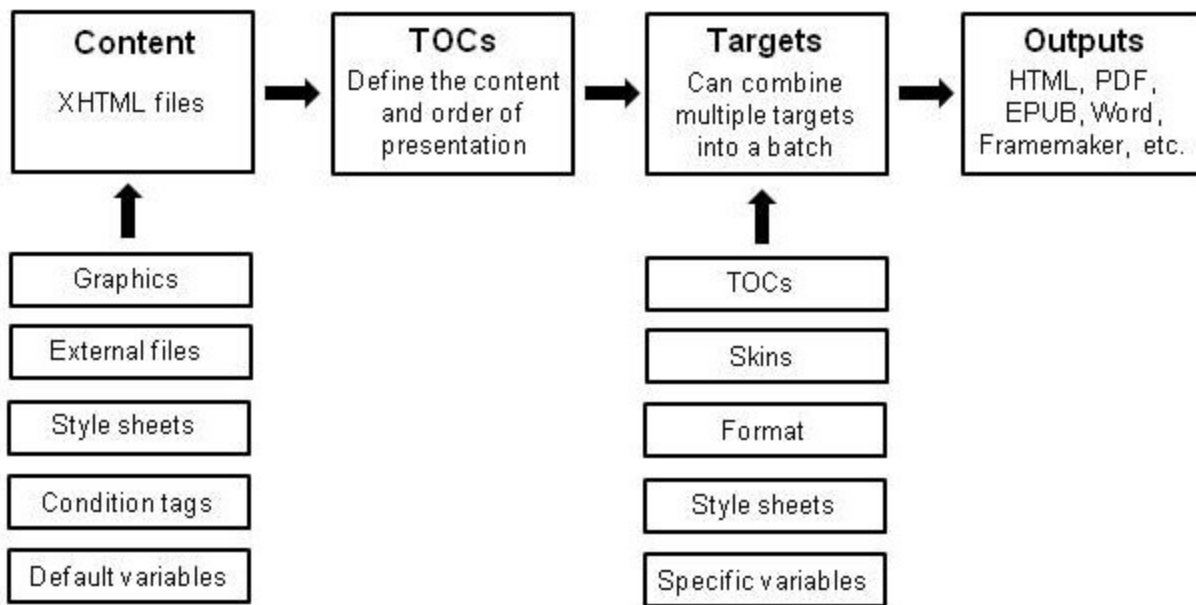
Figure 2: The Home page using the top-navigation (HTML_FRAMELESS) skin



Flare Workflow

The workflow within Flare differs from the traditional workflow employed in most word-processing applications. Rather than writing content as a continuous book (as in Word) or chapter (as in FrameMaker), Flare allows you to create content as a series of topics which you then assemble, by building a table of contents, to create your outputs. Although this workflow can be cumbersome at times, it lends itself well to single-sourcing, by allowing you to pick and choose the content you want to use to create as many different outputs as you require from a single set of source files.

Figure 3: The Flare workflow



Content

You create the content used to build your books as a series of topics. Each topic represents a single HTML page, or h1 or h2 heading in the print output. For example, the topic you are currently reading is a single XHTML file in the project content. For each chapter in a book there is also one level-1 heading topic that serves as the first topic in the chapter.

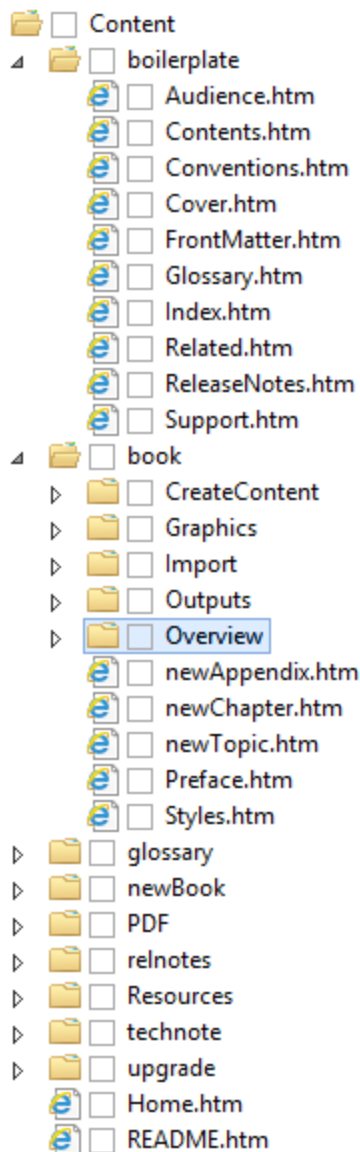
The topics are stored in a hierarchical structure that arranges the topics into folders representing book and chapters, to make the topics easy to find and update. The contents are sorted alphabetically and do not, however, reflect the precise structure of the output books; this is defined by the table of contents used to build the outputs.

Content that is common to all books and can be reused in all of your books is contained in the **bolierplate** folder.

In addition to the XHTML topics you create using Flare's editor, the Contents folder also contains the following:

- > CSS stylesheet used to define the format of the paragraph styles used in the template. This is stored in the Resources/Stylesheets folder
- > any content you want to include in the project that is created in another application, such as PDF files. In this template workflow, the PDF books you generate are published to the PDF folder so that they can be linked from the Home (splash) page.
- > graphic files used in the project. These are stored in the Resources/Images folder.
- > the default variables for the project. The default values are overwritten by target-specific values that you specify in the output targets.

Figure 4: Structure of the contents folder



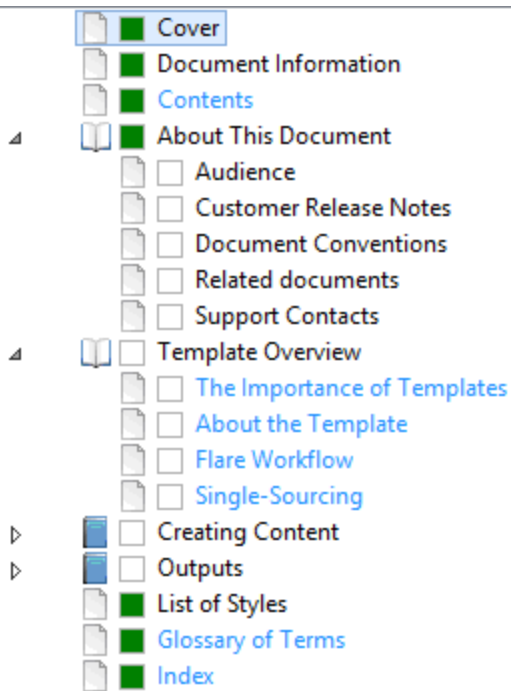
See ["Creating Content" on page 1](#) for more information.

Tables of Contents

In Flare, tables of contents are not built automatically, based on the structure of your document, as is the case with a traditional word processing application, such as Microsoft Word. Instead, Flare allows you to build your books by selecting the topics you want to include in the book from the project contents and arrange them as required to create the book. This methodology, while being somewhat cumbersome, allows you to build multiple outputs from the same set of source files, ideal for single sourcing,

Although this method provides flexibility, it also can lead to an unorganized contents folder, where topics are arranged in a flat file structure, making individual topics difficult to find and update, due to the lack of context. For this reason, it is strongly recommended that you structure your contents as illustrated in ["Structure of the contents folder" above](#).

Figure 5: Structure of the TOC for a book



Targets

Targets specify the format of an output and the components you want to use to build the output. For example, for a single-sourced project containing several books, you would create a PDF target for each book and a single HTML target for the entire project.

When you create a target, you specify the following:

- > the output format of the target. This template uses HTML5 for the online outputs and PDF for the print outputs.
- > the TOC to use when generating the target
- > the stylesheet to use when generating the output. This template uses only one stylesheet for all output formats with stylesheet mediums for print and online to allow for a different look and feel between the online and print outputs. The target also allows you to specify the stylesheet medium to use for the output.
- > the conditional text to include/exclude from the output.
- > the target-specific variable definitions, such as title and part number.
- > the folder(s) to which to write and publish the outputs.
- > several other options which are unused in this template.

You can also create batch targets that allow you to generate multiple outputs from a single command. This template uses batch targets to generate all of the outputs into a single integrated package.

See ["Configuring Your Output Targets" on page 1](#) for more information.

Outputs

The outputs generated by this template are combined into a single integrated package that includes all of the documentation in both HTML and PDF formats. This package can be deployed on-product, to a web server, file server, or DVD/CD with no manual intervention required.

See ["Generating Outputs" on page 1](#) for more information.

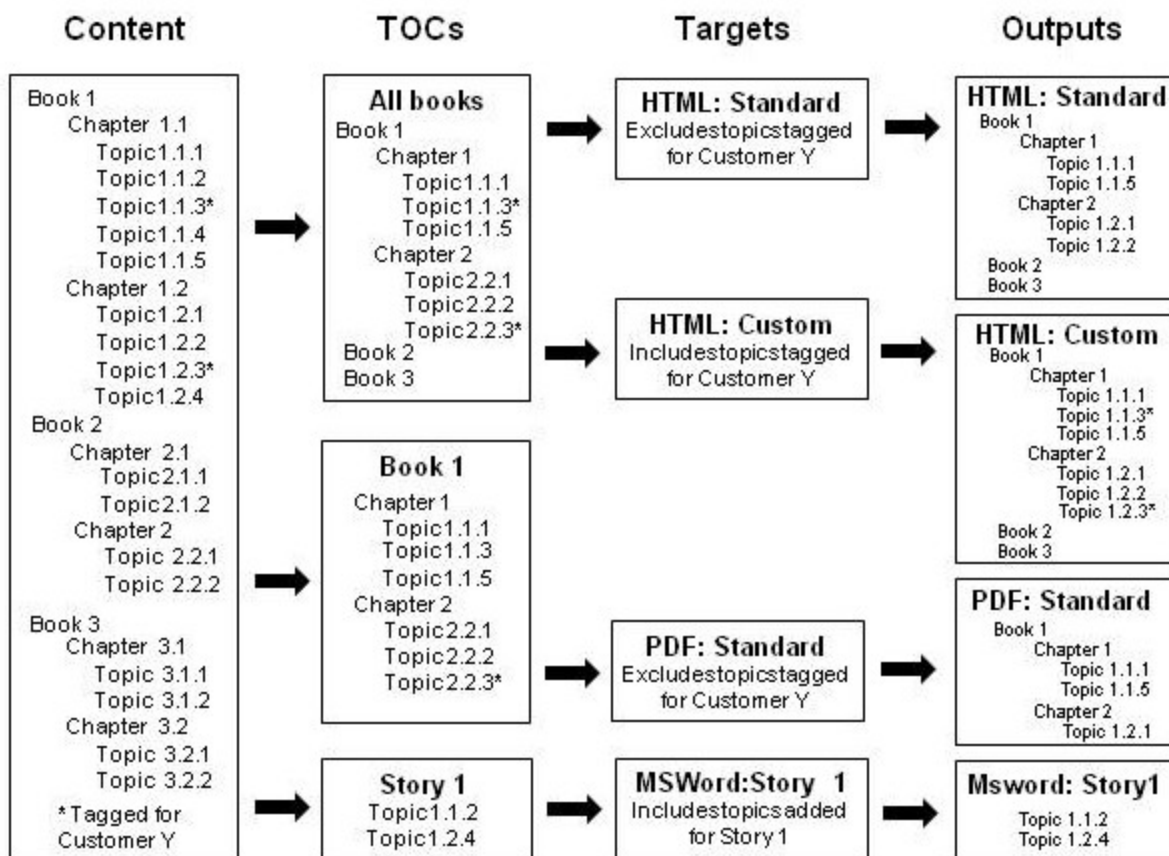
Single-Sourcing

Documentation single-sourcing is the ability to re-purpose a single set of content files to publish outputs in multiple different formats, or customize the outputs for a specific audience or customer by including or excluding material as required. For example, you can use single-sourcing to create outputs in PDF, HTML, Microsoft Word, or EPUB formats from the same set of source files, all from within the Flare application. If you have special content for a specific customer or audience, you can create multiple versions of the same output that include or exclude content, down to the level of a single word, as required to meet the requirements of the intended audience. In addition, you can use single sourcing to rebrand the documents, for example in a reseller arrangement, to include different branding.

How This Template Uses Single Sourcing

This template has been designed for single-sourcing. You create a single set of content files and use TOCs and targets to output the content into a series of PDF books and a single set of HTML files that contain all of the books in the project. The single-sourcing workflow is illustrated in the following figure.

Figure 6: Single-sourcing workflow



In this example, the project consists of three books, delivered to customers as individual PDFs and as a package of HTML files that contains all three books. The books have two different audiences - a standard set is delivered to most customers, while a customized set containing documentation for custom features requested by customer Y is produced for customer Y. The next release of the product will include documentation for Jira story 1 (topics 1.1.2 and 1.2.4). To facilitate reviews, an additional MSWord output is generated that contains these two topics only, for distribution to developers/testers so they can easily comment on the documentation for these new features.



NOTE For most efficient project management, it is recommended that you create a TOC for each PDF book and then reuse these TOCs to build the TOC for the multi-book HTML output, as detailed in [Defining the Output Contents](#).

CHAPTER 2: Creating Content

This is a standard XML documentation template, with some customizations. It consists primarily of headings, paragraphs, lists, and tables. Many other styles are also present, mostly for print objects such as headers, footers, TOCs, etc. All of the styles used in the template are described in the following sections:

- > ["Authoring Best Practices" below](#)
- > ["Creating a New Book" below](#)
- > ["Creating Chapters" on page 23](#)
- > ["Headings" on page 24](#)
- > ["Paragraphs" on page 26](#)
- > ["Notes, Cautions, Warnings, and Tips" on page 27](#)
- > ["Examples and Command and Code Snippets" on page 31](#)
- > ["Lists" on page 40](#)
- > ["Tables" on page 42](#)
- > ["Procedures" on page 48](#)
- > ["Using Variables" on page 52](#)
- > ["Cross-References and Hyperlinks" on page 54](#)
- > ["Creating an Index" on page 55](#)
- > ["Creating a Glossary" on page 56](#)
- > ["Creating the Home Page" on page 57](#)

Authoring Best Practices

This template supports single sourcing. This requires the use of CSS mediums, one for print (PDF) output and one for non-print (HTML) output. When you are authoring, the interface allows you to choose the page layout and medium you want to use. Because the formatting of the PDF outputs are the most complex, it is strongly recommended that you author your documents with Layout set to Print, and Medium set to Print. This will ensure that you see all of the formatting options (such as autonumbers) and that the PDF files you generate from the project will be formatted correctly.

Creating a New Book

The template includes a folder, called **newBook**, that contains all of the elements required to create a new book. When you create a new project, place a copy of the **newBook** folder into your Contents folder and copy it as required to create books within the project.

Structuring Your Content

All of the content for a project is stored in the Contents folder. It is important that you structure the content in a consistent, logical manner to ensure that the information is easy to find and update and closely reflects the structure of the tables of contents used to build the project outputs. The projects created using this template are divided into a series of books. A book is a separate PDF document. In the HTML output, a book is represented by a top-level folder in the HTML table of contents.

Books are represented by a single folder in the project's Content folder. All of the content for the book, except for the graphics, are contained in this folder. Within the book folder are a series of chapter folders, each containing two or more topic files. Each chapter must have an h1 introductory topic file, and one or more h2-level topic files.

The boilerplate Folder

The **boilerplate** folder contains boilerplate material that is common to all documents and can be reused in all of your document TOCs to create the cover, frontmatter, preface, and other elements such as the index and glossary, if used. The files in this folder contain static boilerplate content or use proxies to generate content automatically (for example, the TOC or glossary). If you required different versions of these files (for example for different audiences), you can create copies and place them in different folders or give them different names.

The **boilerplate** folder contains the following files:

Audience.htm	Describes the audience for the book. Although it contains a boilerplate audience definition for a typical Gemalto user, you should edit or amend the definition for your particular product.
Contents.htm	The Contents.htm file is preformatted with the h1.contents style and the TOC proxy. Use it to create a printed table of contents.
Conventions.htm	Describes the Gemalto document conventions for command syntax, documenting UI elements, and using notes, cautions and warnings. All writers need to be familiar with, and adhere to, these conventions. Do not edit this file.
Cover.htm	The Cover.htm file is preformatted with the p.product and p.title styles and the Product and Title variables. Use it to create the document cover.
Glossary.htm	The Glossary.htm file is preformatted with the h1.appendix style and the Glossary proxy. Use it to create a printed glossary as an appendix in a single-document project. For multi-book projects, use the Glossary folder to create a Glossary book.
Index.htm	The Index.htm file is preformatted with the h1.index style and the Index proxy. Use it to create a printed index.
Related.htm	Lists the set of documents related to the current document. You must edit this document to list the related documents.
ReleaseNotes.htm	Describes the purpose of the release notes and provides a link to the release notes on the Gemalto support site.
Support.htm	Lists the support contact information. Body text is from the support_contacts.htm snippet.

The newBook Folder

The **newBook** folder contains the following folders and files, which, along with the boilerplate material, you can use to create new books:

newAppendix	The newAppendix.htm file is preformatted with the h1.appendix style. Use it to create the first page of an appendix. Use the newTopic.htm file to add topics to the appendix if necessary.
newChapter	The newChapter.htm file is preformatted with the h1.chapter style. Use it to create the first page of an chapter. Use the newTopic.htm file to add topics to the chapter if necessary.
FrontMatter.htm	The FrontMatter.htm file is preformatted with the boilerplate material used in the frontmatter. This boilerplate material is contained in the disclaimer.flslnp and trademarks.flslnp snippets. The Document Information heading is conditionalized as an h4 for the print output and a h2 for the online output, so that it can be cross-referenced (from the Preface/preface.htm file). This document must be updated to include revision information for the book.
Preface.htm	The h1 topic for the preface. The default heading for this topic is "About this document". It includes a brief description of the document and a list of cross-references to each chapter and appendix in the document. You must edit this file when creating a new book or adding or removing a new chapter or appendix.

Building the Book

You build book by creating TOCs and targets, as described in ["TOCs: Defining the Output Contents, Structure, and Layout" on page 66](#) and ["Configuring Your Output Targets" on page 73](#).

Creating Chapters

Chapters are used to group a set of topics under a common heading. Chapters provide a brief description of the information contained in the chapter followed by a list (using cross-references) of each of the topics in the chapter, and perhaps overview/concepts information. Chapters begin with the <h1.chapter> tag. The contents of a chapter are contained within a single folder in the project's Contents folder.

To create a new chapter

1. Copy the **newChapter.htm** file to the folder that contains the content for the book to which you want to add the new chapter.
2. Rename the file to reflect the chapter's content.



TIP To ensure problem-free output, do not include spaces or special characters in the file name.

3. Open the file and enter a new heading and a brief description of the content of the chapter.
4. Add a cross-reference to each topic in the chapter.

Headings

Headings are used to begin chapters and topics and to break topics into logical sections. This template supports 5 levels of headings, which are described in detail in the following sections. `<h1>` is a chapter heading. `<h2>` is a major heading, and corresponds to a topic. All topics, with the exception of the chapter heading, begin with an `<h2>` heading. The heading for this section is an `<h2>`. A topic contains all of the information from the start of one `<h2>` until the start of the next `<h2>`. Except for their .ntoc classes, h1, h2, and h3 headings appear in the TOC - h4 and h5 headings do not.

Refer to ["List of Styles" on page 84](#) for information detailing how heading styles are formatted in the print and online outputs.



NOTE Do not use punctuation in headings, such as a colon in an h5 used to introduce a procedure. This is to avoid having the punctuation appear in any cross-references to the heading. Besides that, it is just bad form.

Level-1 Headings

Level-1 headings begin with the h1 tag. They are used to head each chapter and major section of a book. The level-1 headings are also used in the running header in the print outputs.



NOTE In the previous version of this template, you were required to insert a hard return when using the h1.chapter, h1.preface, and h1.appendix styles to force the autonumber to appear above the heading. This restriction has been removed in this release, with the autonumber appearing beside the heading. As a result you must remove any existing manual breaks from these heading styles. See the *Template Upgrade Guide* for details.

There are several h1 classes, as follows:

h1	Typically not used in books, but can be used to break up technical notes or release notes into major sections. In print outputs, automatically inserts a page break. The ability to use a generic h1 is new in this release.
h1.appendix	Used for appendices. For print outputs, adds the autonumber A:APPENDIX {A+}:
h1.chapter	Used for chapters. For print outputs, adds the autonumber CHAPTER {chapnum}:
h1.contents	Used for the table of contents in the print output. Adds the autonumber CONTENTS.
h1.index	Used for the index in the print output. Adds the autonumber INDEX.
h1.preface	Use this class for the preface. For print outputs, adds the autonumber PREFACE:. Recommended preface heading is "About the <title>".

Level-2 Headings

Level-2 headings begin with the h2 tag. They are used to head each topic (**.htm** file) in a chapter. There are several h2 classes, as follows:

h2	Use for topic headings that you want to include in the TOC and that do not force a page break in the printed output.
h2.ntoc	Use for topic headings that you do not want to include in the printed TOC.
h2.pageBreak	Use for topic headings that force a page break in the printed output. Typically used for individual commands in a command reference manual.

Level-3 Headings

Level-3 headings begin with the h3 tag. They are used for subtopics within a topic (**.htm** file). Avoid using an h3 as a heading for a topic - they are typically embedded within an h2 topic. If you have h3 topics, that is typically an indication that you have too many topics and should consider merging the h3 topics into an h2 topic. There are two h3 classes, as follows:

h3	Use for h3 headings that you want to include in the TOC.
h3.ntoc	Use for h3 headings that you do not want to appear in the TOC.

Level-4 Headings

Level-4 headings use the h4 tag. They are used for subsections of an h3 subtopic (**.htm** file) and do not appear in the TOC. Never use an h4 as a heading for a topic file.

Level 5 Headings

Level-5 headings use the h5 tag. They are used for procedure headings and do not appear in the TOC. Never use an h5 as a heading for a topic. Procedure headings always begin with the phrase "To", for example:

To illustrate how level-5 headings are used to introduce a procedure

1. The first step of the procedure.
2. The second step of the procedure.
3. The third, and final, step of the procedure.

Generic Subclasses

You can apply a generic class to any paragraph or heading style. For example, to insert a page break before a paragraph, select the paragraph and then use the Styles drop-down to apply the **.pagebreak** generic class.

.ntoc	Use this class to exclude an h1, h2, or h3 from the table of contents.
.pagebreak	Use this class to insert a page break above the selected paragraph.

Inserting Headings Into a Topic

A level-1 heading is added automatically when you use **newChapter.htm** or **newAppendix.htm** to create a new chapter or appendix. A level-2 heading is added automatically when you use **newTopic.htm** to add a topic to a chapter. You can use the styles window to add level-3 and level-4 headings to a topic.

To insert an <h3> or <h4> heading

1. Place the cursor where you want to insert the heading.
2. Select the **h3** or **h4** style from the **Styles** menu.
3. Type your heading.

Paragraphs

Paragraphs use the <p> tag. This template defines many style classes for the <p> tag, only some of which you will use regularly. The paragraph styles used in day-to-day authoring are used to create paragraphs in text, lists, table cells, or notifications. The template also defines many other style classes that are used for special print-based formats, such as the cover page and headers and footers. These style classes are not used in day-to-day authoring.

Refer to ["List of Styles" on page 84](#) for information detailing how paragraph styles are formatted in the print and online outputs.

Paragraph Style Classes Used to Create Paragraphs in Text

Paragraphs in text, such as this one, use the <p> style.

Paragraph Style Classes Used to Create Lists

Style classes are defined for creating the following list types:

- > Ordered lists. The style classes used to create ordered lists belong to the <p.ol...> style class family.
- > Unordered lists. The style classes used to create ordered lists belong to the <p.ul...> style class family.

Lists are described in detail, with examples, in ["Lists" on page 40](#).

Paragraph Style Classes Used in Table Cells

This template does not use the <td> and <th> tags for table cells, due to their lack of flexibility. Instead, the <p.td> and <p.th> tags are used to format the text within the <td> and <th> elements. See ["Tables" on page 42](#) for more information.

Paragraph Style Classes Used for Examples and Code Samples

You can use the <pre> tag to add preformatted text for examples and code samples. When you use the <pre> tag, the text is inserted as is, with all formatting retained. You can also apply syntax highlighting to preformatted code samples. See ["Examples and Command and Code Snippets" on page 31](#) for more information.

Generic Subclasses

You can apply a generic class to any paragraph or heading style. For example, to insert a page break before a paragraph, select the paragraph and then use the Styles drop-down to apply the **.pagebreak** generic class.

.ntoc	Use this class to exclude an h1, h2, or h3 from the table of contents.
.pagebreak	Use this class to insert a page break above the selected paragraph.

Notes, Cautions, Warnings, and Tips

Notes, cautions, warnings, and tips use different formatting to draw the readers attention to their content. Simple notes, cautions, warnings, and tips that contain only a single paragraph use paragraph styles. More complex notes, cautions, warnings, and tips that consist of multiple paragraphs and lists are formatted using a div.

Notification formats

See the following topics for examples of the various notification formats:

- > ["Notes" below](#)
- > ["Cautions" on the next page](#)
- > ["Warnings" on the next page](#)
- > ["Tips" on page 29](#)

Creating notifications

The following procedures describe how to create notifications:

- > ["Inserting a single-paragraph notification using a paragraph style" on page 29](#)
- > ["Inserting a multi-paragraph notification using a div style" on page 30](#)

Notes

Notes are used to highlight important information that helps the user to successfully complete a task or understand an important concept.

Single-paragraph notes

The following is a single paragraph note. It uses the **p.note** style.



NOTE Take note. Contains important or helpful information.

Multi-paragraph notes

The following is a multi-paragraph note. It is formatted using standard paragraph styles and then converted to a note using the **div.note** tag.

NOTE Take note. The following steps contain important or helpful information:

1. This is the first step you need to perform.
2. This is the second step you need to perform.
 - Heed this point.
 - Also understand the impact of this point.

This completes the note. Thanks for reading.

Cautions

Cautions are used to highlight important information that may help prevent unexpected results or data loss.

Single-paragraph cautions

The following is a single paragraph caution. It uses the **p.caution** style.



CAUTION! Exercise caution. Contains important information that may help prevent unexpected results or data loss.

Multi-paragraph cautions

The following is a multi-paragraph caution. It is formatted using standard paragraph styles and then converted to a caution using the **div.caution** tag.

CAUTION! Exercise caution. The following important information may help prevent unexpected results or data loss:

> This is a p.ul within a caution div.

This is a paragraph within a caution div.

1. This is a p.ol1_start within a caution div.
2. This is a p.ol1 within a caution div.

Warnings

Warnings are used to highlight information that alerts the user to the potential for catastrophic data loss or personal injury.

Single-paragraph warnings

The following is a single paragraph warning. It uses the **p.warning** style.



****WARNING**** Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.

Multi-paragraph warnings

The following is a multi-paragraph tip. It is formatted using standard paragraph styles and then converted to a tip using the **div.warning** tag.

****WARNING**** Be extremely careful and obey the following safety and security measures to avoid catastrophic data loss or personal injury:

- > Do not run with scissors.
- > Do not use the hair dryer while in the bathtub.
- > Do not fly a kite next to power lines.

Tips

Tips are used to highlight information that helps to complete a task more efficiently, such as a best practice.

Single-paragraph tips

The following is a single paragraph tip. It uses the **p.tip** style.



TIP This is some information that will allow you to complete your task more efficiently.

Multi-paragraph tips

The following is a multi-paragraph tip. It is formatted using standard paragraph styles and then converted to a tip using the **div.tip** tag.

TIP This is some information that will allow you to complete your task more efficiently. It consists of the following steps:

1. The first thing you need to do is perform this step.
2. Then do this.
3. Lastly, you want to do this.

Now wasn't that helpful!

Inserting a single-paragraph notification using a paragraph style

Use the following paragraph styles to insert notifications:

Note	p.note
Caution	p.caution
Warning	p.warning
Tip	p.tip

To insert a single-paragraph notification using a paragraph style

1. Place your cursor where you want to insert the notification. You can insert notifications in body text, lists, or tables.
2. Select the paragraph style for the notification you want to create. The notification icon, label, and shading are inserted automatically.
3. Type the text for the notification.

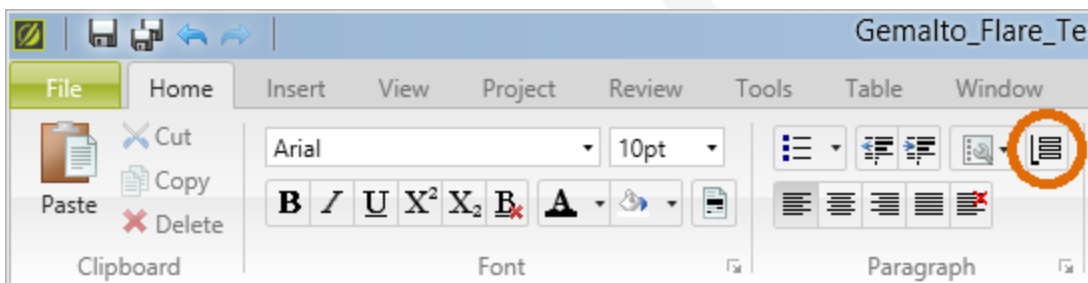
Inserting a multi-paragraph notification using a div style

Use the following div styles to format multiple paragraphs as a notification:

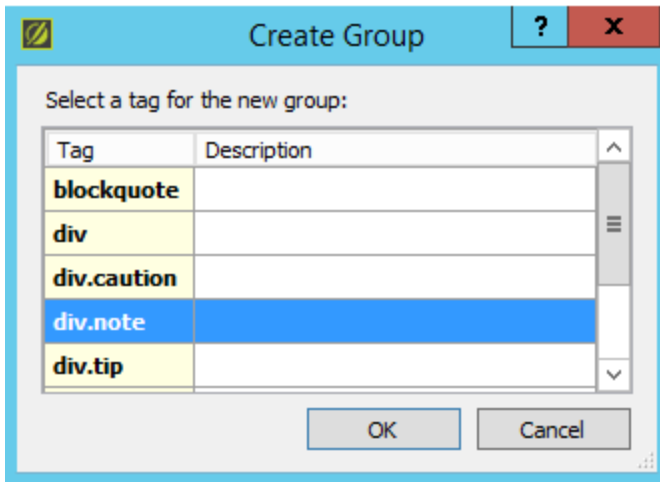
Note	div.note
Caution	div.caution
Warning	div.warning
Tip	div.tip

To insert a multi-paragraph notification using a div style

1. Type your notification using standard paragraph styles.
2. Select the paragraphs you want to include in the notification.
3. Group the paragraphs using the **Group** function.



The **Create Group** dialog is displayed:



4. Select the div for the type of notification you want, and click **OK**.

Examples and Command and Code Snippets

When documenting a command line interface (CLI) or application programming interface (API) it is often necessary to include command output examples or code snippets. These examples or snippets are usually written in a monospace font and formatted using manual methods such as tabs and spaces.

General highlighting

You can use `div.highlight` to highlight any information you like, as shown below:

Example of text highlighted with `div.highlight`



NOTE The top margin is removed for the first child of the div (heading or paragraph style, etc) so that the margin is consistent regardless of the style applied to the first paragraph or heading. The heading above is an h3.

HSMs provide an ultra-secure vault for your digital keys. Integrate your applications to authenticate to an HSM, and protect:

- > PKI key generation & storage (online and offline CA keys)
- > SSL/TLS
- > Code Signing
- > Certificate Signing & Validation
- > Document signing
- > HSMaaS – Private & Public Cloud Environment
- > Transaction processing
- > Database encryption
- > Smart card issuance
- > Hardware root of trust for the Internet of Things (IoT)
- > Blockchain
- > Compliance including GDPR, PCI-DSS, HIPAA, eIDAS, and more

Preformatted text

When including these examples and snippets in the documentation, it is important to retain the original formatting and monospace font, and to be able to import these examples and snippets without the need for reformatting. This requirement is satisfied by the HTML `<pre>` tag, which is also available as a paragraph format in Flare. Additional `<pre.>` tags are provided that add indents for examples inserted in lists. See ["Using Preformatted Text" on the next page](#) for details.

Syntax highlighting

Although preformatted text allows for improved readability of code samples, most developers use tools that use both indentation and syntax highlighting to enhance code readability. Syntax highlighting tools automatically add color to specific code elements, allowing the elements to be easily distinguished from each other. Syntax highlighters are smart, and are able to detect the code language, and highlight accordingly. Some examples of code authoring tools that provide syntax highlighting are Notepad++ and Eclipse. This template integrates the Prism syntax highlighter (<http://prismjs.com/>), which allows you to apply a code-specific character tag to your preformatted text that triggers the Prism JavaScript to use syntax highlighting for the specified code type. See ["Using Syntax Highlighting for Code Samples" below](#).

Using Preformatted Text

You can use the `<pre>` tag to add preformatted text. When you use the `<pre>` tag, the text is inserted as is, with all formatting retained. This is ideal for capturing screen output.

Examples within body text

The following example uses the `<pre>` style:

```
lunacm:> partition contents
```

```
The User is currently logged in. Looking for objects in the
User's partition.
```

```
Number objects: 2
Handle: 7      Label: Known
Handle: 8      Label: Generated DES3 Key
```

```
Command Result : No Error
```

Examples within lists

The following examples show how to use the `<pre.listLevel1>` and `<pre.listLevel2>` styles:

- > This is a level 1 unordered list item `<p.ul1>`.

```
This is an example inserted within a level1 list item <pre.listLevel1>
The User is currently logged in. Looking for objects in the User's partition.
```

- > Another level 1 unordered list item `<p.ul1>`.

- This is a level 2 unordered list item `<p.ul2>`.

```
This is an example inserted within a level1 list item <pre.listLevel2>
The User is currently logged in. Looking for objects in the User's partition.
```

Using Syntax Highlighting for Code Samples

This template optionally integrates the Prism syntax highlighter (<http://prismjs.com>), which allows you to apply a code-specific character tag to your preformatted text that triggers the Prism JavaScript to use syntax highlighting for the specified code type. Prism integration is disabled by default, and must be enabled if you want to use it, as described in ["Enabling Prism integration" on the next page](#).

If your documentation uses code samples, you can apply syntax highlighting to the code samples to make them easier to read, using the same syntax highlighting that developers are familiar with when working with code in tools such as Eclipse or Notepad++.



NOTE Syntax highlighting only works in HTML outputs, since it uses JavaScript to interpret and highlight the text. Syntax highlighting does not work in PDF outputs. You will not see the syntax highlighting when working in Flare.



CAUTION! Because this output relies on Javascript, ensure that the Javascript is not stripped out when transferring a compiled project. For example, when transferring a project through email within Gemalto, you must encrypt the email before sending.

Enabling Prism integration

Syntax highlighting is implemented by integrating the Prism syntax highlighter as follows:

- > The **prism.js** and **prism.css** files were generated at <http://prismjs.com/> and saved to the <project>\Content\Resources\PlugIns\SyntaxHilite\Prism directory.
- > The following lines were added to body section of the HTML master pages (<project>\Content\Resources\MasterPages\Default.flmsp, FRAMELESS.flmsp, FRAMELESS_TOOLBAR.flmsp and FRAMELESS_MENU.flmsp).

```
<!-- Uncomment the following line to enable syntax highlighting
<script type="text/javascript" src="../../PlugIns/SyntaxHilite/Prism/prism.js"></script>
-->
```

- > The following lines were added to the top of the CSS file (**Gemalto_Template.css**):

```
/* If you want to use syntax highlighting, uncomment the following line: */
/* @import url('../../PlugIns/SyntaxHilite/Prism/prism.css'); */
```

To enable Prism integration

1. Open the master page that you will use for your HTML outputs and uncomment the line that enables the **prism.js** script:

```
<!-- Uncomment the following line to enable syntax highlighting -->
<!-- <script type="text/javascript" src="../../PlugIns/SyntaxHilite/Prism/prism.js"></script> -->
```

So that it looks like this:

```
<!-- Uncomment the following line to enable syntax highlighting -->
<script type="text/javascript" src="../../PlugIns/SyntaxHilite/Prism/prism.js"></script>
```



TIP If you open the file in Flare's internal text editor (right-click on the file and select **Open With > Internal Text Editor**), comments are displayed in green.

2. Open the CSS file (**Resources/Stylesheets/Gemalto_Template.css**) and uncomment the line that enables the prism.css file:

```
/* If you want to use syntax highlighting, uncomment the following line: */
/* @import url('../../PlugIns/SyntaxHilite/Prism/prism.css'); */
```

So that it looks like this:

```
/* If you want to use syntax highlighting, uncomment the following line: */
@import url('../PlugIns/SyntaxHilite/Prism/prism.css');
```

Applying syntax highlighting to your code samples

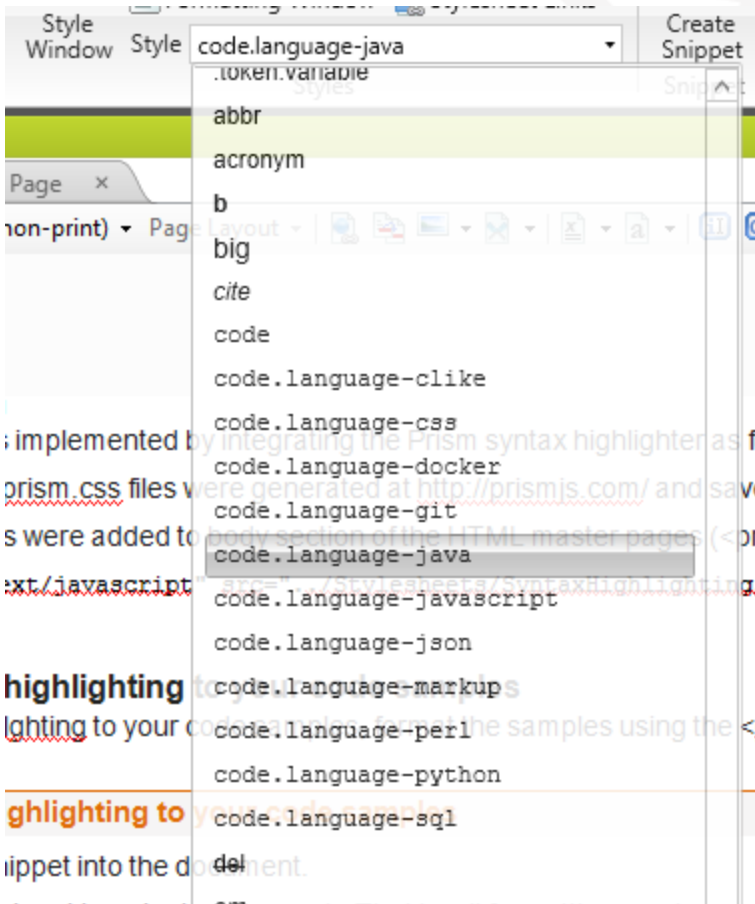
To apply syntax highlighting to your code samples, format the samples using the `<pre>` tag, and then apply the desired **code.language-`<language>`** character style.

To apply syntax highlighting to your code samples

1. Insert the code snippet into the document.
2. Ensure that the snippet is a single paragraph. That is, all formatting must use spaces/tabs and soft returns (Shift-Enter). You can edit the source using the Text Editor mode to remove the `<p>` and `</p>` tags if they were added when you pasted the snippet.
3. Apply the **pre** style to the code snippet paragraph.
4. Highlight the contents of the paragraph (do not include the paragraph markers). The **Style** field will display **(text)** to indicate that you are in character style mode.



5. Use the Style menu drop-down to select and apply the desired **code.language-`<language>`** character style to the highlighted text.



Examples

The following code samples illustrate syntax highlighting for the various supported **code.language-`<language>`** styles.

C-like

The following code snippet uses the **code.language-clike** style.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <errno.h>

char* program_name;

void system_error(char* cause, int exit_code)
{
    fprintf(stderr,"%s: %s: %s\n",program_name,cause, (char*) strerror(errno));
    exit(exit_code);
}

int main(int argc, char **argv)
{
    if(argc != 2)
    {
        printf("Usage: %s <filename>\n",argv[0]);
        exit(EXIT_FAILURE);
    }
    program_name = argv[0];
    /* Permissions for the new file */
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;

    /* filename for the new file */
    char* filename = argv[1];

    /* Create a new file */
    int fd = open(filename,O_CREAT | O_EXCL,mode);
    if(fd == -1)
    {
        system_error("open",EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);
}
```

CSS

The following code snippet uses the **code.language-css** style.

```
/*<meta />*/
@import url('../SyntaxHighlighting/Prism/prism.css');

body
{
    font-family: Arial;
    margin-bottom: 0px;
```

```

margin-left: 0pt;
margin-right: 0px;
margin-top: 0px;
}

h1
{
  page-break-before: always;
  font-family: Arial;
  margin-top: 0;
  margin-left: 0pt;
  margin-bottom: 60pt;
  text-align: left;
  mc-hyphenate: never;
  font-style: normal;
  color: #e06c08;
  font-size: 18pt;
  font-weight: 500;
}

```

Docker

The following code snippet uses the **code.language-docker** style.

```

# Build
  IMAGEID=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 32 | head -n 1) && sudo docker
build -t $IMAGEID .

# Run new container
  CONTAINER_ID=$(sudo docker run -t -d -p 0.0.0.0:81:80 $IMAGEID)

# Login into running container. REQUIRES DOCKER >= 1.3
  sudo docker exec -it $CONTAINER_ID bash

# Container stdout
  sudo docker logs $CONTAINER_ID

# Inspect container config
  sudo docker inspect $CONTAINER_ID

# Stop container
  sudo docker stop $CONTAINER_ID

```

Git

The following code snippet uses the **code.language-git** style.

```

$ git push origin amend-my-name

Counting objects: 34, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (21/21), done.
Writing objects: 100% (28/28), 6.87 KiB, done.
Total 28 (delta 13), reused 12 (delta 7)
To git@github.com:evildmp/afraid-to-commit.git
* [new branch]      amend-my-name -> amend-my-name

```

Java

The following code snippet uses the **code.language-java** style.

```
Jaxenter out = null;
try {
    out = new Jaxenter (new FileWriter("filename", true));
    out.write("aString");
} catch (IOException e) {
    // error processing code
} finally {
    if (out != null) {
        out.close();
    }
}
```

Javascript

The following code snippet uses the **code.language-javascript** style.

```
var Prism = require('prismjs');
// The code snippet you want to highlight, as a string
var code = "var data = 1;";
// Returns a highlighted HTML string
var html = Prism.highlight(code, Prism.languages.javascript);
{
    "hsms": "/api/lunasa/hsms",
    "syslog": "/api/lunasa/syslog",
    "ssh": "/api/lunasa/ssh",
    "network": "/api/lunasa/network",
    "services": "/api/lunasa/services",
    "actions": "/api/lunasa/actions",
    "ntp": "/api/lunasa/ntp",
    "forceSoLogin": false,
    "version": "6.2.0-6",
    "time": "/api/lunasa/time",
    "snmp": "/api/lunasa/snmp",
    "webServer": "/api/lunasa/webServer",
    "ntls": "/api/lunasa/ntls",
    "ssh": "/api/lunasa/ssh",
    "sensors": "/api/lunasa/sensors"
}
```

JSON

The following code snippet uses the **code.language-json** style.

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
```

```

    "vOffset": 250,
    "alignment": "center"
  },
  "text": {
    "data": "Click Here",
    "size": 36,
    "style": "bold",
    "name": "text1",
    "hOffset": 250,
    "vOffset": 100,
    "alignment": "center",
    "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
  }
}

```

Markup

The following code snippet uses the **code.language-markup** style.

It's very easy to make some words **bold** and other words *italic* with Markdown. You can even [link to Google!](http://google.com)

Perl

The following code snippet uses the **code.language-perl** style.

```

my $filename = 'check.txt';
if(open F,">$filename")
{
    close F;
    unlink "$filename";
    print 'Permissions are OK.';
}
else
{ print 'Permission denied for writing files!' }

```

Python

The following code snippet uses the **code.language-python** style.

```

# This program adds two numbers

num1 = 1.5
num2 = 6.3

# Add two numbers
sum = float(num1) + float(num2)

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))

```

SQL

The following code snippet uses the **code.language-sql** style.

```

mysql> CREATE PROCEDURE myProc()
-> BEGIN
->     DECLARE i INT DEFAULT 1;
->

```

```

->      SET autocommit=0;
->
->      DROP TABLE IF EXISTS test_table ;
->      CREATE TABLE test_table (
->          id          INT PRIMARY KEY,
->          some_data VARCHAR(30))
->      ENGINE=innodb;
->
-> END$$
Query OK, 0 rows affected (0.00 sec)

```

Lists

This template supports three levels of ordered lists and three levels of unordered lists.

Ordered Lists

Ordered lists are created using the following styles that you use to maintain correct indentation of your list items and any paragraphs or tables they contain. The standard ol, li structure is not used due to its lack of flexibility:

List level	List items	Embedded paragraphs	Embedded tables
Level 1	p.ol1Start p.ol1	p.listLevel1	List1
Level 2	p.ol2Start p.ol2	p.listLevel2	List2
Level 3	p.ol3Start p.ol3	p.listLevel3	List3



NOTE Do not use the quick list feature of Flare to create ordered lists.

Example of an ordered list

1. This is the first item in a level1 ordered list <p.ol1Start>.

This is a paragraph inserted within a level1 list item <p.listLevel1>, followed by a **List1** table:

Option 1	This table uses the List1 table style.
Option 2	This table uses the List1 table style.

2. This is a level 1 ordered list item <p.ol1>.

- a. This is the first item in a level2 ordered list <p.ol2Start>. Let's see what happens when we add enough text to make the paragraph wrap to the next line. Cool. It wraps as expected.

This is a paragraph inserted within a level2 list item <p.listLevel2>, followed by a **List2** table:

Option 1	This table uses the List2 table style.
-----------------	--

Option 2	This table uses the List2 table style.
-----------------	--

3. This is a level1 ordered list item <p.ol1>.

a. This is the first item in a level2 ordered list item <p.ol2>.

b. This is a level2 ordered list item <p.ol2>.

i. This is the first item in a level3 ordered list <p.ol3Start>.

ii. This is a level3 ordered list item <p.ol3>.

This is a paragraph inserted within a level3 list item <p.listLevel3>, followed by a List3 table:

Option 1	This table uses the List3 table style.
-----------------	--

Option 2	This table uses the List3 table style.
-----------------	--

iii. This is a level3 ordered list item <p.ol3>.



NOTE Notes, cautions, and warnings align with 3rd-level lists, since they use the maximum indentation.

4. This is a level 1 ordered list item <p.ol1>.

Unordered Lists

Unordered lists are created using the following styles that you use to maintain correct indentation of your list items and any paragraphs or tables they contain. The standard ul, li structure is not used due to its lack of flexibility.



NOTE Do not use the quick list feature of Flare to create unordered lists.

List level	List items	Embedded paragraphs	Embedded tables
Level 1	p.ol1Start p.ol1	p.listLevel1	List1
Level 2	p.ol2Start p.ol2	p.listLevel2	List2
Level 3	p.ol3Start p.ol3	p.listLevel3	List3

An example of an unordered list

> This is the first item in a level1 unordered list <p.ul1>.

This is a paragraph inserted within a level1 list item <p.listLevel1>, followed by a **List1** table:

Option 1	This table uses the List1 table style. Width=97%
Option 2	This table uses the List1 table style. Width=97%

> This is a level 1 ordered list item <p.ol1>.

- This is the first item in a level2 unordered list <p.ul2>. Let's see what happens when we add enough text to make the paragraph wrap to the next line. Cool. It wraps as expected.

This is a paragraph inserted within a level2 list item <p.listLevel2>, followed by a **List2** table:

Option 1	This table uses the List2 table style. Width=94%
Option 2	This table uses the List2 table style. Width=94%

> This is a level1 unordered list item <p.ul1>.

- This is the first item in a level2 unordered list item <p.ul2>.
- This is a level2 unordered list item <p.ul2>.
 - This is the first item in a level3 unordered list <p.ul3>.
 - This is a level3 unordered list item <p.ul3>.

This is a paragraph inserted within a level3 list item <p.listLevel3>, followed by a List3 table:

Option 1	This table uses the List3 table style. Width=91%
Option 2	This table uses the List3 table style. Width=91%

- This is a level3 ordered list item <p.ul3>.



NOTE Notes, cautions, and warnings align with 3rd-level lists, since they use the maximum indentation.

> This is a level 1 unordered list item <p.ul1>.

Tables

The format of the tables supported table in this template are defined by the following table styles. These formats are defined in **Content/Resources/TableStyles/<style>.css**. If you need custom shading, or need to manipulate tables in other ways, you can use the Table editor to change the default settings.

Tables With Borders

Tables with borders consist of an optional shaded header row and unshaded table text rows. When used without a header, the table styles can be used for definition lists, for example to define a series of fields in a dialog. All of the table styles look the same. The only difference is the amount of left indent.

Page	Used for page-width tables. This style does not have a left indent.
List1	Used for tables inserted within a level1 list. This style has an 15pt left indent.
List2	Used for tables inserted within a level2 list. This style has a 30pt left indent.
List3	Used for tables inserted within a level3 list. This style has a 45pt left indent.

Tables Without Borders

Tables without borders have no borders between cells or rows. These table formats are typically used to float graphics. To avoid clutter, only a page-width borderless table is included by default. You can use **Table > New Table Style** to add list-level borderless styles if required.

Page_no_borders	Used for page-width tables. This style does not have a left indent.
------------------------	---

The following example illustrates a borderless table used to float graphics:

SafeNet Luna Network HSM



Paragraph Formats Supported in Table Cells

When you add a table, it will use the <td> and <th> tags. To use paragraph tags in the cells, just press Return to add a paragraph and select the paragraph styles you want to use. The following paragraph tags are supported in tables:

- > th
- > td
- > p
- > p.ul1
- > p.ul2
- > p.ul3
- > p.ol1, p.ol1Start
- > p.ol2, p.ol2Start
- > p.ol3, p.ol3Start
- > p.note
- > p.caution

- > p.warning
- > p.tip



NOTE Adding paragraphs and list items to table cells is managed using complex selectors. Complex selectors add context-sensitivity to styles, so that paragraphs and list items that are embedded within a `<td>` tag use the formatting specified by the complex selector. Complex selectors are defined in the CSS using the syntax `<parent_tag> <embedded_tag>`. For example, the style `<td p>` defines how a paragraph added to a table cell is formatted. Complex selectors can only be defined/edited using a text editor. Flare does not provide a GUI.

Using Tables in Body Text

Use the **Page** style for page-width tables. You can format the table with or without a caption and/or heading row, as illustrated in the following examples.

Table With a Caption and Heading Row

The following table includes a caption and a heading row.

Table 1: This is a standard page-width table with a heading row and caption `<caption>`

Table heading. Uses the <code><th></code> style	Table heading. Uses the <code><th></code> style
Table text. Uses the <code><td></code> style.	Table text. Uses the <code><td p></code> style. Notice that <code><td></code> and <code><td p></code> look exactly the same.

Table heading. Uses the <th> style

Table text. Uses the <td> style.

Paragraph in cell. Uses <th p> style. When you press enter after a <td>, a paragraph is inserted after the original <td> and another paragraph is inserted into the cell. The <td p> This style can be changed by editing Page.css using a text editor.

1. Start of ordered list in table <td p.ol1Start>.
2. Ordered list item in table <td p.ol1>
 - a. Start of level 2 ordered list in table <td p.ol2Start>
 - b. Level 2 ordered list item in table <td p.ol2>
 - i. Start of level3 ordered list in table <td p.ol3Start>
 - ii. Level3 ordered list item in table <td p.ol3>
- > Unordered list item in table <td p.ul1>
 - Level 2 unordered list item in table <td p.ul2>
 - Level 3 unordered list item in table <td p.ul2>



NOTE Note in a table <td p.note>



CAUTION! Caution in a table <td p.caution>

Table heading. Uses the <th> style

Table text. Uses the <td> style.

Paragraph in cell. Uses <th p> style. When you press enter after a <td>, a paragraph is inserted after the original <td> and another paragraph is inserted into the cell. The <td p> This style can be changed by editing BookStyles.css using a text editor.

1. Start of ordered list in table <td p.ol1Start>.
2. Ordered list item in table <td p.ol1>
 - a. Start of level 2 ordered list in table <td p.ol2Start>
 - b. Level 2 ordered list item in table <td p.ol2>
 - i. Start of level3 ordered list in table <td p.ol3Start>
 - ii. Level3 ordered list item in table <td p.ol3>
- > Unordered list item in table <td p.ul1>
 - Level 2 unordered list item in table <td p.ul2>
 - Level 3 unordered list item in table <td p.ul2>



****WARNING**** Warning in a table <td p.warning>



TIP Tip in a table <td p.tip>

Table Without a Caption

Most tables do not need a caption, so don't clutter your documentation with captions if they are not required.

Table heading. Uses the <th> style**Table heading. Uses the <th> style**

Table text. Uses the <td> style.

Table text. Uses the <td p> style. Notice that <td> and <td p> look exactly the same.

Table Without a Heading Row

When used without a heading row, you can use tables to describe a list of options, for example.

Option 1	Use this option to enable option 1 behavior.
Option 2	Use this option to enable option 2 behavior.

Using Tables in Lists

Use the **List1** and **List2** table styles to insert tables into level-1 and level-2 lists, respectively. Tables are typically inserted into lists to describe a list of options.

Level-1 List Tables

Level-1 list tables have an 15 pt left indent, as illustrated in the following example. To accommodate for the indent, set the width to 97%:

- > This is a level-1 list item.

Table style	List1
Autofit behavior	Autofit to window: 97%
Align	Default

Level-2 List Tables

Level-2 list tables have a 30 pt left indent, as illustrated in the following example. To accommodate for the indent, set the width to 94%:

- > This is a level-1 list item.
 - This is a level-2 list item

Table style	List2
Autofit behavior	Autofit to window: 94%
Align	Default

Level-3 List Tables

Level-2 list tables have a 45 pt left indent, as illustrated in the following example. To accommodate for the indent, set the width to 91%:

- > This is a level-1 list item.
 - This is a level-2 list item
 - This is a level-3 list item

Table style	List3
Autofit behavior	Autofit to window: 91%
Align	Default

Inserting Tables

Use the **Insert Table** dialog to insert tables.

To insert a page table

1. Select **Insert > Table**. Don't use the quick method (down arrow). Click on the **Table** icon instead to open the **Insert Table** dialog.
2. Specify the number of rows, columns and header rows. Footer rows are not supported.
3. Specify a caption, if desired. Ensure that all of the fields except **Text** are set to (default).
4. In the **Autofit Behavior** section, select **Autofit to Window** and set it to 100%.
5. Ensure that **Align** is set to **(default)**. If set to any other value, the settings in the table style will not be respected.
6. Select the **Page** table style.
7. Click **OK**.

To insert a level-1 list table

1. Select **Insert > Table**. Don't use the quick method (down arrow). Click on the **Table** icon instead to open the **Insert Table** dialog.
2. Specify the number of rows, columns and header rows. Footer rows are not supported.
3. Specify a caption, if desired. Ensure that all of the fields except **Text** are set to (default).
4. In the **Autofit Behavior** section, select **Autofit to Window** and set it to **97%**. This prevents the table from bleeding over the right margin in the HTML output.
5. Ensure that **Align** is set to **(default)**. If set to any other value, the settings in the table style will not be respected.
6. Select the **List1** table style.
7. Click **OK**.

To insert a level-2 list table

1. Select **Insert > Table**. Don't use the quick method (down arrow). Click on the **Table** icon instead to open the **Insert Table** dialog.
2. Specify the number of rows, columns and header rows. Footer rows are not supported.
3. Specify a caption, if desired. Ensure that all of the fields except **Text** are set to (default).
4. In the **Autofit Behavior** section, select **Autofit to Window** and set it to **94%**. This prevents the table from bleeding over the right margin in the HTML output.
5. Ensure that **Align** is set to **(default)**. If set to any other value, the settings in the table style will not be respected.
6. Select the **List2** table style.
7. Click **OK**.

To insert a level-3 list table

1. Select **Insert > Table**. Don't use the quick method (down arrow). Click on the **Table** icon instead to open the **Insert Table** dialog.

2. Specify the number of rows, columns and header rows. Footer rows are not supported.
3. Specify a caption, if desired. Ensure that all of the fields except **Text** are set to (default).
4. In the **Autofit Behavior** section, select **Autofit to Window** and set it to **91%**. This prevents the table from bleeding over the right margin in the HTML output.
5. Ensure that **Align** is set to **(default)**. If set to any other value, the settings in the table style will not be respected.
6. Select the **List3** table style.
7. Click **OK**.

Procedures

Much of the content within task-based technical documents consists of procedural information. This is likely also the content that is most often used by your readers, since they refer to the documentation when they do not know how to perform a task. For this reason, it is important that all procedures look the same, and use the same structure.

Creating a Procedure

Use ordered lists to document procedures (a series of steps that describe how to perform a specific task). All procedures are introduced with an `<h5>` in the form 'To <task to be performed>', as illustrated below.



NOTE Do not follow your procedure heading with a colon (:). If you do, any cross-reference to the procedure heading will include the colon, which will look strange.

To document a procedure `<h5>`

1. Start the list `<p.ol1Start>`.
2. Add a step `<p.ol1>`.
 - a. Start of a series of sub-substeps `<p.ol2Start>`.
 - b. Another sub-substep `<p.ol2>`.
 - c. This substep provides a series of options, which are best presented in a table format: `<p.ol2>`

Table style	List2.
Table width	94%

3. Another step `<p.ol1>`. This step requires that you add a paragraph.
This is a paragraph inserted in a level-1 list. `<p.listLevel1>`
4. Specify values for the following fields in the dialog: `<p.ol1>`

Field 1 <td>	A description of the field. This items has several sub-items:<p> <ol style="list-style-type: none">1. This is the first item.<p.ol1Start>2. This is the second item. It has a couple of sub-items:<p.ol1> <ol style="list-style-type: none">a. This is a sub-item.<p.ol2Start>b. Another sub-item. <p.ol2>3. This item has a few additional points:<p.ol1> <ul style="list-style-type: none">• point 1 <p.ul2>• point 2 <p.ul2>
Field 2 <td>	A description of the field.<td>
Field 3 <td>	A description of the field.<td>
Field 4 <td>	A description of the field.<td>

4. When you perform this step the following things happen:<p.ol1>
 - this happens<p.ul2>
 - this also happens<p.ul2>
 - this happens too<p.ul2>
5. This is the end of the procedure.<p.ol1>

Example Procedure

This is an example of a real SafeNet procedure that has been converted to use the styles defined in this template. The original procedure is at <http://sentinelldk.safenet-inc.com/LDKdocs/WebHelp/CreateEnt.htm>.

To create an Entitlement

1. (Optional) In the **Customer** field, specify the Customer for whom the Entitlement is being created. Type the first few characters of the Customer name. A list of matching names is populated. Select a name from the list or click **Add New** to create a new Customer. To perform an advanced search, click .
If the Customer has multiple contact e-mail IDs, you can choose one from the E-mail field. You can also leave this field blank.
2. (Optional) In the **Channel Partner** field, specify a Channel Partner. Type the first few characters of the Channel Partner name. A list of matching names is populated. Select a name from the list. To perform an advanced search, click .
If the Channel Partner has multiple contact e-mail IDs, you can choose one from the **E-mail** field.
3. Provide basic information for the Entitlement.
 - a. In the **Ref ID 1** and **Ref ID 2** fields you can enter information that identify the Entitlement in another system. For example, the order code in your company's ERP system. Adding information is optional.
 - b. Enter **Start Date** and **End Date** for the Entitlement.
To extend the end date indefinitely, select **Never Expires**.

4. To select the Products to be included in the Entitlement, on the Product Details pane click **Actions > Add**. The Product Selection pop-up appears.
5. Select a Product, and click **OK**. The selected Product is added to the Entitlement. You can add multiple Products to an Entitlement.



NOTE The Product specifications must be compatible with the Entitlement type. For example, you cannot add a Product that must be locked to a Sentinel HL key to an Entitlement for Sentinel SL keys.

- If the license terms have been defined for all the Features in the Product, the details are not displayed. You can click on the left of the Product to view the details.
- If the license term values for a Feature have not been defined, all the Feature details for the Product are displayed. The License Terms for Features with values that need to be defined are shown in orange. After the values are defined the License Terms are shown in green.
- Configure rehosting for Products for which rehosting needs to be specified during Entitlement creation—for such Products, a drop-down list appears in the Rehost column. Select a value from the following:

Enable	Enable rehosting for the Product.
Disable	Disable rehosting for the Product.
Leave as it is	Retain the value of rehost as is in the Protection key.

- If you want to exclude a Feature from the Product, select the **Exclude** check box. Only Features that were specified as Excludable at the time the Product was created can be excluded.
 - Select a Product and click **Memory** to display the Memory pop-up. For more information about editing memory data, see Editing Memory Data during Entitlement Generation.
6. For each Feature for which license term values have not been defined, click to select the row and click **Actions > Configure** to display the Configure License Terms pop-up. You can also multi-select Features (by selecting multiple rows) that require the same licensing details.

Alternatively, to configure individual Features click the orange link in the License Terms column.

Specify the required values for this Entitlement. For detailed information on specifying license terms, see Defining License Terms.



NOTE The Entitlement can be added to the production queue only after the license term values have been defined for all the Features in all the Products included in the order.

7. Specify the Entitlement type from one of the following:

Product Key	Associates the Entitlement items to one or more Product Keys.
Hardware Key	Writes the Entitlement items to one or more Sentinel HL keys.
Protection Key Update	Enables changes to be made to the license data stored in deployed keys.

8. Save as Draft, Queue, or Produce the Entitlement:

Save Draft	Save the Entitlement in Draft state until it is ready to be added to the production queue.
Queue	Make the Entitlement available for production.
Produce	Produce the Entitlement immediately.

Figures and Graphics

Figures are used to express information graphically. They can be inserted with or without a caption. Typically graphics used to illustrate a concept include a caption, while screen captures used in a procedure do not.

Storing your Graphic Images

Store all of the images associated with a project in the **Content > Resources > Images** folder. For very large projects, you can create sub-folders in the **Images** folder, if necessary, to organize the images.

Graphic Formats

There are currently no technical documentation graphic standards at Gemalto. The following recommended formats are based on some preliminary studies by the Ottawa group.

Table 1: Recommended graphic formats

Graphic type	Recommended format
Bitmap (screen captures)	JPG or PNG
Vector (line drawings)	SVG

Vector Graphics

For vector graphics, the open-source tool **Inkscape** is recommended. Inkscape is the open-source equivalent of Adobe Illustrator, and allows you to create SVG (scalable vector graphics) files that you can edit in Inkscape and insert directly into your Flare documents. Once you embed an SVG graphic in a Flare topic, you can simply right-click on it and select **Open Linked File With** to open and edit the SVG in Inkscape. SVGs also scale automatically without any loss in resolution, making them the best choice for vector (line) drawings.

Sizing Your Graphics

To ensure that your graphics fit within the 180 mm (7.09 inch) page width of the PDF page, scale all of your graphics so that they are no wider than 7.09in x 96pixel/in = 680 pixels.

GIMP is a good free open-source graphics editing tool that you can use to resize bitmap images. For best results with screen captures, resize your bitmaps using Sinc (Lanczos3) interpolation (Image > Scale Image > Quality).

Inserting Figures

You can insert a figure with or without a caption. Unlike table captions, which are specified in the **Insert Table** dialog, figure captions are added as a paragraph (**p.figureCaption**) above the figure.

To insert a figure

1. Use the **p.figureCaption** style to enter a caption for the figure, if necessary.
2. Select **Insert > Image**.
The **Insert Image** dialog is displayed.
3. Select the image and choose the resize options as required.
4. Enter a screen tip, if desired.

Using Variables

The use of variables for common terms that appear in the documentation is strongly encouraged. Using variables for terms such as the product name and product release allows you to update this information in a single location and have it apply to all occurrences of the variable that appear in the document.

This template uses a single variable set (Default) that contains default settings for all of the variables used in a project. You can override the default settings for a specific target by setting the variable values for the target in the Variables tab of the Target Editor.

The Default Variable Set

The default variable set contains all the variables that are used in the project. Note that the variables need not apply to all of the documents in the project. The default variable set is located at **Project/Variables/Default.flvar**.



NOTE You typically override the default settings for a specific target by setting the variable values for the target in the **Variables** tab of the **Target Editor**. You can, however, also set specific variables in the **Default.flvar** file if they apply to all documents in the project. For example, you may do this for the Release and Revision variables.



CAUTION! Flare gives you the ability to create variable sets other than the default set. There is no need to do this, however, since you can override the default variable values at the target level. If you do create custom variable sets you will also need to create new page layouts with footers that reference the new custom variables. This can quickly get out of hand, and is strongly discouraged.

Table 1: Default variables

Variable	Definition	Default Setting
CompanyName	Specifies the common name for the company.	Gemalto

Variable	Definition	Default Setting
CompanyNameLong	Holdover from the SafeNet days, retained for backwards compatibility. For Gemalto, the company name is Gemalto. No long or short.	Gemalto
Copyright	Adds the text "Copyright" and the current year (yyyy).	"Copyright" yyyy
DateLong	Automatically adds the current date in the form dd MMMM yyyy.	dd MMMM yyyy
DateShort	Automatically adds the current date in the form MMMM yyyy. Used in the footer.	MMMM yyyy
FirmwareVersion	Specifies the firmware version.	<FirmwareVersion>
PartNumber	Specifies the document part number. Since variables are defined at the target level, you can assign a unique part number to each individual PDF document in the project. The HTML output (which includes all documents) can also be assigned a part number. Used in the footer and frontmatter.	<PartNumber>
Product	Specifies the product name. Used on the cover and in the footer.	<Product>
ProjectTitle	Specifies the title of the project for multi-book projects. For example, Luna HSM Product Documentation.	<ProjectTitle>
Release	Specifies the product release, for example Release 1.0. Used in the footer and frontmatter.	<Release>
Revision	Specifies the document revision, for example Rev. A. Used in the footer and frontmatter.	<Revision>
SubjectNotes	Specifies the subject in technical notes. Used in the technical note footer.	<SubjectNotes>
Title	Specifies the document title. Used on the cover and in the footer.	<Title>



NOTE The default variable set lists all of the required variables used in the template (for example, in the headers and footers, on the cover page, etc). You may find that you need additional variables for your specific projects. If you need additional variables, think about them carefully and avoid adding excessive variables since too many variables can be difficult to manage. Once you add a variable, it can be difficult to remove at a later date if it is no longer required.

Cross-References and Hyperlinks

Cross-references and hyperlinks serve as navigation aids to help readers quickly and easily find related information.

When creating links, always use cross-references if possible. Cross-references are preferred over hyperlinks for the following reasons:

- > they automatically update when you change the source.
- > in print outputs, they include the page number for the reference.

Cross-References

Flare allows you to create cross-references to headings and bookmarks within the current topic or other topics within the project. When cross-referencing headings other than the top-level heading in other topics, you need to click on the Bookmark icon. Headings (h1, h2 and h3) are automatically bookmarked. If you want to cross-reference to a paragraph other than a heading, you need to bookmark it first.

Cross-Reference Formats

This template provides the following cross-reference formats:

Cross-reference format name	Output format	Cross-reference format	Examples
Madcap:xref	PDF	{paratext} {pageref}	See " Cross-References " above
	HTML	{paratext}	
Madcap:xref.ParaNum	PDF and HTML	{paranumonly}	This is a cross-reference to step 2 (the xref_Step2 bookmark) in the procedure " To insert a cross-reference " below.

Inserting Cross-References

Use the **Insert Cross-Reference** dialog to insert a cross-reference.

To insert a cross-reference

1. Press **CTRL-Shift R**, or use the GUI, to open the **Insert Cross-Reference** dialog.
2. Select either **Topic in project** or **Place in this document**, as relevant, in the **Link to:** field.
3. Navigate to the topic of heading you want to reference.
4. Click **OK** to insert the cross-reference.

Hyperlinks

Use hyperlinks only where it is not possible to use a cross-reference. A example of a legitimate use for a hyperlink is on the **Home.htm** page, where the PDF icons and document titles use hyperlinks to link to a specific document.

Inserting Hyperlinks

Use the **Insert Cross-Reference** dialog to insert a cross-reference.

To insert a hyperlink

1. Press **CTRL-K**, or use the GUI, to open the **Insert Hyperlink** dialog.
2. Configure the link as required and click **OK** to insert the link.

Creating an Index

Flare provides multiple methods for creating an index for your project. The index is accessed from the Index tab in HTML outputs, which provides a project-wide index. A book-specific index is created for each PDF output, and is included in the Index chapter for the book.



TIP In the age of Google and online documentation, search engines have all but replaced indexes. Indexes are time-consuming to create and maintain, and unless they are complete and up-to-date are not particularly useful. It is recommended that you apply search engine optimization best practices when authoring your content rather than creating and maintaining indexes.

Entering Index Items

Flare provides multiple methods for creating an index for your project. Refer to the Flare documentation to find the method best suited to you.

Including an Index in a PDF Book

When Flare generates an output, it collates all of the index entries found in the files included in the TOC for the output and includes them in the index. For PDF files, the index is printed using the Index proxy, which is part of the **Index.htm** file in the template.

To include an index in a book

1. Copy the **Index.htm** file from the BookTemplate folder into the folder for the book.
2. Include the **Index.htm** file in the TOC for the book.

Creating a Glossary

Flare provides a GUI for creating a glossary of terms for your project. The GUI allows you to enter the terms and definitions you want to include in your glossary without having to worry about the order in which they appear. Depending on your project output formats and whether your project contains a single document or multiple documents, you can generate the glossary in the following formats:

- > as an HTML file that is accessed by clicking on the Glossary tab in the HTML output.
- > as an appendix in the PDF output of a single-book project
- > as a separate book in the PDF output of a multi-book project

Refer to the Flare documentation for details on how to use the glossary feature.

Entering Glossary Terms and Definitions

Use the Glossary Editor to enter the terms and definitions you want to include in the glossary for your project.

To populate the glossary

1. From the Project Organizer, go to **Project > Glossaries** and open the glossary for your project. Flare provides a default glossary called **MyGlossary** which you can use to create the glossary, or you can import an existing glossary or copy and rename the default **MyGlossary** file if desired.

The glossary is opened in the Glossary Editor.

2. Use the Glossary Editor to enter your glossary terms and definitions.

Including the Glossary as an Appendix in a Single-Book Project

When Flare generates an output, it collates all of the glossary entries found in the glossary associated with the project and includes them in the glossary. For PDF files, the glossary is printed using the Glossary proxy, which is part of the **Glossary.htm** file in the template.

To include a glossary as an appendix in a book

1. Copy the **Glossary.htm** file from the BookTemplate folder into the folder for the book.
2. Include the **Glossary.htm** file in the TOC for the book.

Including the Glossary as a Separate Book in a Multiple-Book Project

When Flare generates an output, it collates all of the glossary entries found in the glossary associated with the project and includes them in the glossary. For PDF files, the glossary is printed using the Glossary proxy, which is part of the **Glossary.htm** file in the template.

To include an glossary as a separate book in a project

1. Copy the **Glossary** folder from the template **Contents** folder into the **Contents** folder for the project.
2. Include the contents of the **Glossary** folder in the TOC for the project.

Creating the Home Page

The Home page lists all of documents in the project and provides links to each document in HTML and PDF formats. The Home page is the startup page for the project target, and is displayed when a user clicks on the **Home** link in the breadcrumbs. It is also the first item in the table of contents in the HTML TOC.

The Home page is a standard Flare topic located at the root of the Contents folder. If your project contains multiple products or product versions requiring different targets, you can maintain different home pages for each target, or use conditional text, as desired.

Figure 7: The Home page



To create the Home page

1. Copy the **Home.htm** file from the template's Contents folder to your project's Contents folder.
2. Edit the file to add a title and description for each document in the output.
3. Add a PDF icon for each book (**Resources/Images/icons/pdf_icon.png**), if necessary.
4. Create hyperlinks for each HTML link:
 - link books to the first page of the preface (<book>/Preface/preface.htm)
 - link notes to the note (eg, note,htm) or the first topic in the note if you break the note into a series of files.
5. Create hyperlinks for each PDF file:
 - a. Generate a PDF for each book. Ensure they are named appropriately. The PDF file name is the same as the name of the target used to generate it.

- b. Place the PDFs in the **Contents/PDF** folder
- c. Create a hyperlink for each PDF from the PDF icon associated with the PDF. Include a screen tip for the hyperlink in the form "Download a PDF of the <bookname>". The screen tip is displayed when the user mouses over the PDF icon.

After initially creating the hyperlink, you can replace the PDFs in the **Contents/PDF** folder with updated versions, and as long as the PDF files have the same name as the originals (used to create the hyperlinks), the hyperlinks will remain intact. Each time you build the project, you can publish the PDFs to the **Contents/PDF** folder as part of the build process (see "[Creating the PDF Targets](#)" on page 74), without needing to edit the home page to recreate the hyperlinks.



TIP To create a shaded row in the table, you must first merge the cells in the row and then shade the merged cell.

- 6. If you are using top-navigation outputs using the HTML_FRAMELESS skins, edit the home page topic properties to use the HOME master page. The HOME master page does not include the menu or toolbar, which do not work on the home page, and cause formatting issues. See [HTML Top-Navigation Outputs](#) for more information:
 - a. Right-click on the home page topic file in the Content Explorer and select **Properties**.
 - b. Set **Topic Properties > Masterpage:** to HOME

CHAPTER 3: Importing Content

You may have been creating documentation in other applications, such as Microsoft Word or Adobe FrameMaker, but are now ready to migrate your documents to Flare. Madcap wants you to migrate, and provides pretty good wizards to help you import the content from your legacy documents into a Flare project. Since FrameMaker import is relatively straightforward and similar to Word import, this chapter covers importing Word documents only (see ["Importing MS Word Documents" on the next page](#)). Importing is covered in detail in the Flare documentation, although much of the information does not apply when importing to the Gemalto Flare template.

When you import a legacy document into a Flare project, you import only the **content** and the **structure** of the original document (along with some local formatting, such as bolded text). All of the formatting elements (paragraph and character styles, headers and footers, cover page and frontmatter, and some old boilerplate text) are left behind and replaced by formatting elements in the Flare template (CSS, page layouts, TOCs, etc.) Some formatting elements, like paragraph and character styles, are mapped to the Flare styles on import. Other formatting elements, such as page layouts, you configure after importing the content. Graphics and other non-text content comes through cleanly.



TIP It is far easier to clean up the source prior to import than it is to clean it up in Flare after import. Although you may be tempted to jump right in, planning how you import the content so that it works when delivered as individual topics has a major impact on the useability and maintainability of the documentation once in Flare. If you are new to Flare, read the entire *Gemalto Flare Template User Guide* to familiarize yourself with Gemalto's Flare implementation before proceeding.

Topic-Based Authoring

When you import your single-file Word document, or several-chapter FrameMaker document into Flare, it's going to be "chunked" into multiple individual HTML files, or topics. You can configure how the document is broken into topics by specifying which heading levels will mark the start of a new topic. Typically, you would create topics for h1 and h2 headings, such that h1 topics would contain the h1 and all text up to the next h1 or h2, and h2 topics would contain the h2 and all text up to the next h1 or h2.

In Flare, the topics are threaded together using a TOC, which is separate file that defines the order and structure of the document. The import process creates a default TOC with a flat file structure that retains the order of the original document, but not its structure (ie chapters, page layouts, hierarchy). The same is true for the topics - you get a folder full of topics named using the heading they were created from. There is no arranging of the chapters into folders or any other sort of organization. You will need to organize both the content and TOC of each imported document. In addition, once the topics are imported, you are free to change their contents and structure as required, including the heading level associated with the file when it was imported.

Chunking Strategy

Probably the biggest mistake you can make when importing a Word document into Flare is to create too many files. When chunking your information, think about the context of the individual chunks. They should be able to stand alone, such that the reader, and the author, don't require the context of the topics that are above it or below it in the TOC to understand the context of the topic they are currently viewing or editing. This serves a dual purpose:

- > For the author, this allows you to maintain all of the information required to perform a task in a single file so that you do not need the TOC. For example, place all of the conceptual information for a chapter into a single file (the h1 topic), and the overview and step-by-step procedures for each task in the chapter in its own h2 topic.
- > For the reader, this allows them to quickly "drill down" through the documentation to find the information they need. If they are a new user needing conceptual information, they read the h1 chapter topic. If they understand the concepts and just want to perform a specific task, they'll go directly to the task, which provides an overview of the task, who can perform the task and what they need, and step-by-step procedures for performing the task.

Chunking Methodology

When you import a document, you specify how it is chunked based on the heading levels in the source document. Since the headings in the source document may not necessarily reflect how you want to structure your individual topic files in Flare, you may need to do some retagging to be able to generate standalone topics that contain the information you want.

For example, you may want to combine an h1 section, that provides a series of cross-references to the chapter contents, with the conceptual information in the first two h2 sections of a chapter to create a single overview topic. To do this, you can temporarily retag the h2 sections that you want to embed in the h1 topic to h3, or anything you like, so that they do not get split into separate topics on import. Once the import is complete, you can retag them in Flare. This is much faster than moving the content around once it is in Flare, since once the content is in Flare, it is part of a database. This database maintains a list of the links between all of the files in the project (cross-references, TOC entries, stylesheets, etc). This means that every time you move/rename/delete content, Flare needs to update its database, and ask you for confirmation before it does. This can be time-consuming.

Importing MS Word Documents

You use the Flare Microsoft MS Word import wizard to import Word documents into an existing Flare project based on this template. You import into an existing project because the import process maps your Word styles to the styles defined by the project's CSS. The Flare project you import to must therefore contain the Gemalto CSS (in **Content/Resources/Stylesheets/Gemalto_Template.css**). Importing MS Word documents into Flare is a multi-step process, described in detail in the following sections:

1. ["Preparing Your MS Word Documents for Import" on the next page](#)
2. ["Using the MS Word Import Wizard" on page 62](#)
3. ["Post-Import Cleanup and Configuration in Flare" on page 63](#)

Preparing Your MS Word Documents for Import

To ensure a clean import with a minimal amount of cleanup required once the content is in Flare, you must prepare your document for import. This involves restructuring the content so that it works as individual topics, removing boilerplate and generated content, such as the cover, frontmatter, table of contents, and preface, and removing autonumbering and bullets from lists so that they are imported as paragraphs, and not list items.

To prepare your MS Word documents for import

1. Plan how you want to import your content, as described in ["Importing Content" on page 59](#)
2. Restructure your content into standalone topics, as described in ["Topic-Based Authoring" on page 59](#).
3. Remove the following sections from the front of document. You will replace this content with standard boilerplate or generated content once the content is imported into Flare:
 - the cover
 - the frontmatter (document information, revision history, trademarks, disclaimer)
 - the table of contents
 - the preface (Release Notes, Audience, Document Conventions, Support Contacts)
 - the index
4. Remove the bullets from the unordered list paragraph styles. This allows you to import the list items as paragraphs rather than lists. Otherwise you will need to unbind the lists once in Flare, a very tedious process.
 - a. Open the **Styles** window.
 - b. Select a bulleted list style (these are p_ul1, p_ul2, and p_ul3 if you are migrating from the Gemalto Word template).
 - c. Click on the down arrow and select **Modify**. The **Modify Style** dialog is displayed.
 - d. Click on the **Format** button and select **Numbering**. The **Numbering and Bullets** dialog is displayed.
 - e. Click on the **Bullets** tab and click on **None** in the Bullet Library.
 - f. Click **OK** to save your changes.
 - g. Repeat for all used bulleted list styles.



NOTE This may not work for all of your lists. It is recommended that you go through the document and ensure that you remove the bullets from all unordered lists. You can do this by selecting the lists items and selecting **None** from the bulleted list icon drop-down in the **Paragraph** formatting area.

5. Remove the autonumbers from the ordered list paragraph styles. This allows you to import the list items as paragraphs rather than lists. Otherwise you will need to unbind the lists once in Flare, a very tedious process.
 - a. Open the **Styles** window.
 - b. Select a numbered list style (these are p_ol1, p_ol2, and p_ol3 if you are migrating from the Gemalto Word template, but there are likely more).
 - c. Click on the down arrow and select **Modify**. The **Modify Style** dialog is displayed.

- d. Click on the **Format** button and select **Numbering**. The **Numbering and Bullets** dialog is displayed.
- e. Click on the **Bullets** tab and click on **None** in the Bullet Library.
- f. Click **OK** to save your changes.
- g. Repeat for all used numbered list styles.



NOTE This may not work for all of your lists. It is recommended that you go through the document and ensure that you remove the autonumbers from all ordered lists. You can do this by selecting the lists items and selecting **None** from the numbered list icon drop-down in the **Paragraph** formatting area.

Using the MS Word Import Wizard

The MS Word Import wizard walks you through the import process. You can run the wizard as many times as required to achieve the desired results in Flare.

To import a Word document into an existing Flare project

1. If you are creating a new project (that is, you have not imported any documents yet), take a copy of the Gemalto Flare Template and open it in Flare. Remove any extraneous content if desired. At a minimum you need the newBook folder, the Resources folder and the README file. You probably want to keep the Home file as a template. You can also remove any unused TOC, target, and destination files from the Project folder. You can also rename the project file (Gemalto_Flare_Template.flrprj) as well (you'll have to close Flare to do this).
2. Select **Project > Import > MS Word Documents....** The wizard starts.
3. On the **Import Options** screen, select **Import into this project** and click **Next**.
4. On the **Select MS Word Documents to import** screen, do the following:
 - a. Uncheck the **Link Generated Files to Source Files** checkbox. You want to bring the files in unlinked.
 - b. Click on the **+** to select the files you want to import. You can import a single file or multiple files. If you select multiple files, they will all use the same import mapping rules you define in a subsequent step.
 - c. Click **Next**. Flare will begin scanning the selected document(s) to determine a list of Word styles that are used in the document. Review the list. If there are styles listed that you do not know how to map to a Flare style, cancel, check the Styles defined in your Word document, and retag as required.
5. On the **'New Topic' Styles** screen, specify the styles to want to use to define the start of a new topic by moving them from the **Used Word Styles** column to the **'New Topic' Styles** column. To move a style to another column, select it and click the arrow button. You would typically start new topics for Heading1 (including any appendix styles) and Heading2 topics, but may include additional styles, depending on how you tagged the source document. Click **Next**.
6. On the **Options** screen, accept the defaults, and click **Next**.
7. On the **Stylesheet** screen, do the following:
 - a. Click on the **Stylesheet** button, and select the CSS for the project (<current project>/Content/Resources/Stylesheet/Gemalto_Template.css).
 - b. Select **Don't Preserve MS Word Styles**.



CAUTION! Ensure that you do not bring forward any styles from the source Word document. That is, do not preserve Word styles, or convert inline formatting to CSS styles. If you do, subsequent updates to the template will overwrite these non-supported styles.

8. On the **Paragraph Style Mapping** screen, specify how you want to map the MS Word paragraph styles in the source document to the paragraph styles defined in the Flare CSS. To map a style, select the Word style and the Flare style you want to map it to, and click **Map**. The column on the left is updated to indicate the Flare style that the Word style will be mapped to. Ensure that all of the Word styles are mapped to Flare styles and then click **Next** to proceed.
9. On the **Character Style Mapping** screen, specify how you want to map the MS Word character styles in the source document to the character styles defined in the Flare CSS. To map a style, select the Word style and the Flare style you want to map it to, and click **Map**. The column on the left is updated to indicate the Flare style that the Word style will be mapped to. Any unmapped styles will be ignored on import, such that the text associated with the unmapped character style is imported without formatting. Click **Finish** to begin the import.
10. When the import is complete, the **Accept Imported Documents** dialog is displayed, listing all of the files that were generated during the import process. Generated files include topic files (.htm) and graphic files (.png, .jpg, .svg, etc). A default TOC (.fltoc), master page (.flmsp), and CSS (.css) are also generated. You may want to keep the TOC. You can discard the master page (.flmsp), and CSS (.css) since you will be using the CSS and master page defined in the template.

The dialog allows you to click on a generated file in the **File** list to display a preview of the file in the **Imported** window if the dialog. If you are happy, or just want to look at the results in more detail in Flare, click **Accept** to import the converted content into your Flare project. The following folders/files are added to the project:

- A folder with the same name as the imported document, that contains all of the topics created during the import, is added to the **Content** folder. The topics are added in a flat file structure. There is no hierarchy based on chapters or any other structural elements of the source document.
 - A folder with the same name as the imported document, that contains all of the topics created during the import, is added to the **Content/Resources/Images** folder. The topics are added in a flat file structure. There is no hierarchy based on chapters or any other structural elements of the source document.
 - A master page with the same name as the imported document is added to the **Content/Resources/Images** folder. You can discard this file.
 - A TOC with the same name as the imported document is added to the **Project/TOCs** folder. It retains the structure of the source document.
11. Repeat the import process as many times as required to achieve the desired results.

Post-Import Cleanup and Configuration in Flare

Once you are satisfied with the imported content, you can begin reassembling the document in Flare and cleaning up the content to remove any extraneous metadata, resize graphics, and otherwise fix things so that the content can be used to generate HTML and PDF outputs. You must do your cleanup and configuration in Flare.

To cleanup the imported content and create and configure your TOCs and targets

1. Organize the content into meaningful folders (typically by chapter).
2. Rename the files and directories so that they use short file names and no spaces. For example, if you have a topic named **Backup and Restore HSMs and Partitions.htm**, rename it to something short and descriptive, like **backup.htm**.



CAUTION! The path to any file in your project cannot exceed 256 characters. If you exceed this limit, your builds will fail. There are also path length limits when creating ISOs and other archive files, so be careful. Best practice is to keep things short. Short file and folder names, that do not contain spaces, makes forwarding URLs to your topics easier too.

3. Add the boilerplate content from the template to the folder containing the content for the imported document:
 - a. Copy the **Preface** folder from the **newBook** folder to the folder containing the content for the imported document.
 - b. Copy the **Contents.htm**, **Cover.htm**, and **FrontMatter.htm** files from the **newBook** folder to the folder containing the content for the imported document.
4. Edit the **Preface/Preface.htm** file to add a cross-reference to the h1 topic for each chapter in the document.
5. Edit the TOC for the imported document to add the boilerplate content. You can also restructure the TOC as desired. For example, you can remove any entries that point to a bookmark rather than a file so that your TOC links only to individual topics, not the subheadings they contain. If you do this, you can also remove the TOC bookmarks from the Flare content that were added as part of the import. See ["TOCs: Defining the Output Contents, Structure, and Layout" on page 66](#) for more information about structuring TOCs.
6. Create targets for your desired outputs. See ["Configuring Your Output Targets" on page 73](#) for more information.

CHAPTER 4: Generating Outputs

This template provides several different formats you can use when publishing your content. You assemble and configure your outputs using TOCs and targets and build the targets to generate the outputs. You can use batch targets to generate multiple outputs in one step, and use build scripts to automate the build process.

PDF Outputs

The format of your PDF outputs is defined by the page layouts you apply in the TOC. You can apply page layouts to publish your content as a book or note. See [PDF Outputs](#).

HTML Outputs

The format of your HTML outputs is defined by the skin you specify in the target, as follows:

HTML_FRAMELESS	Displays the TOC for the target at the top of the window. For enhanced navigation, you can apply different master pages to add a Topic Toolbar and/or a Menu proxy. This skin is also well suited to on-product documentation, as it does not use frames, which can trigger security warning in vulnerability scans. See HTML Top-Navigation Outputs
HTML_FRAMES	Displays the TOC for the target in a frame to the left of the content. Because it provides superior navigation, it is recommended that you use the frames skin for large projects. See HTML Tri-Pane Outputs
HTML4	An older format that uses frames. Deprecated.

TOCs

TOCs define the content, and structure of your outputs, and for PDF outputs, the document layout. See ["TOCs: Defining the Output Contents, Structure, and Layout" on the next page](#).

Targets

Targets specify the TOC used to generate the output and format of the outputs. There are many other options you can specify. See ["Configuring Your Output Targets" on page 73](#).

Building Your Outputs

You can build your targets individually or as a batch job that builds multiple targets. You can also automate your builds using the madbuild command. See the following for more information:

- > ["Setting Up Your Project for Combined HTML and PDF Outputs" on page 79](#)
- > ["Building and Publishing Output Targets" on page 80](#)

TOCs: Defining the Output Contents, Structure, and Layout

Flare TOCs are separate files that define the structure of your outputs. TOCs define which topics and folders from your Contents folder appear in the output, the order in which they appear and, for print outputs, page and chapter numbering and page layout attributes. The TOCs for a project are contained in the TOC folder in the Project Organizer. Flare provides a TOC editor for creating and editing your TOCs.

When you create an output target, for example a PDF book, you specify the TOC you want to use to create the target. Flare then assembles the output using the topics defined in the TOC and the attributes defined in the target (format, conditions, variables, etc).



NOTE You can use conditional text to further filter the contents of the output by including or excluding conditionally tagged topics.

PDF TOCs

For PDF outputs, the TOC defines not only the order of the content, but also the page layouts applied to its folders and topics, and the document chapter and page numbering. You use TOCs, for example, to use either the note or book page layouts when creating a document. You can also apply page layouts that add a Draft or Confidential watermark to each page in your document.

Re-using TOCs

After you create a TOC, you can re-use it within another TOC within the project by dragging an existing TOC into a new TOC. This feature allows you to create a single TOC for a book and re-use that TOC within other TOCs created for different targets. For example, if you are creating a project with PDF and HTML outputs, you can create a TOC for each PDF target (book), and then re-use these TOCs to create an multi-book HTML TOC. To re-use a TOC, simply drag it from your Project/TOC folder into the TOC you are creating.



NOTE Best practice is to maintain only a single TOC for a book and re-use to generate all of your outputs, taking advantage of the PrintOnly and ScreenOnly conditions, unless you are using the topic menu proxy, which loads re-used TOCs as single topics (see ["HTML TOC Structure for Frameless Outputs That Use the Topic Menu Proxy" on page 71](#)).

Limitations When Re-using TOCs

Although re-using TOCs is best practice for tri-pane outputs that use the HTML_FRAMELESS skin, there are some limitations which may make it impractical, as follows:

- > In the TOC file, you cannot link an imported TOC to a topic. If you do, when you click on the imported TOC in the output, only the link loads. It does not expand the imported TOC.
- > If you are using the context-sensitive menu proxy with the HTML_FRAMELESS skin, imported TOCs are imported as a single object, which causes undesirable behavior. For example, if an imported TOC is the child of the selected topic, the entire imported TOC is loaded as a child, upsetting the structure of the TOC.

PDF TOCs

You need to create a PDF TOC for each book or note in your project. PDF outputs include the following extra files that are not part of the HTML outputs:

- > **Cover.htm** - the front cover for the PDF book
- > **Contents.htm** - the table of contents for the PDF book

PDF TOCs also define the following print-only attributes:

- > page numbering. You must set the page numbering for the cover to page 1.
- > chapter and appendix numbering. You must set the chapter numbering for the first chapter to 1 and the appendix numbering for the first appendix to A.
- > page layouts. For a book, you must set the page layout and page/chapter breaks for the cover, document information (frontmatter), contents, index, and the first page of each chapter and appendix. For a note, you only need to set the first page of the document to use the Cover page of the Notes page layout.

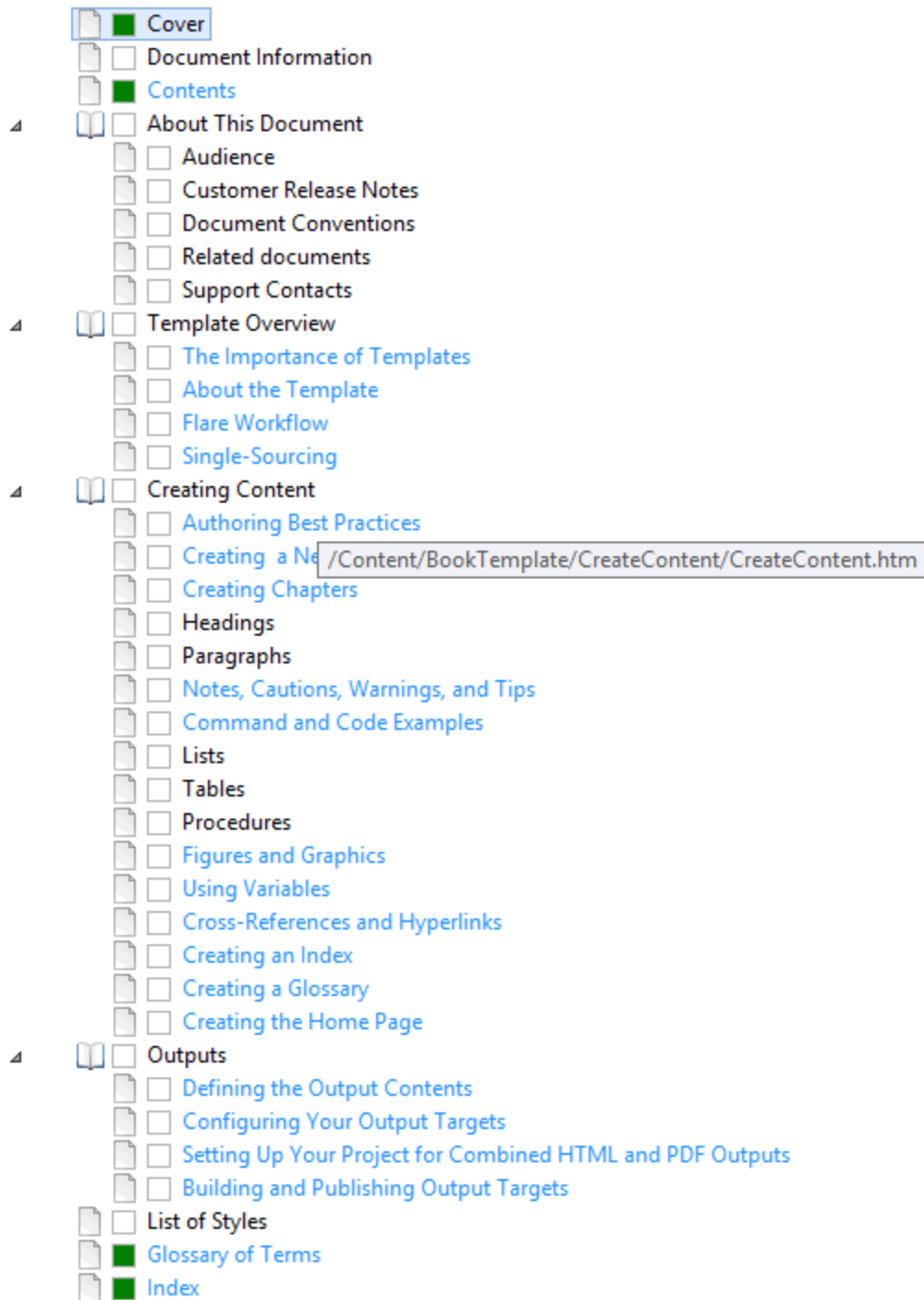
The template includes page layouts that you can use to add a Draft or Confidential watermark to your documents. The watermark page layouts are in Content/Resources/PageLayouts/Watermk. You apply them using the Printed Output tab of the TOC Properties dialog for a topic, as detailed in the following procedures.



NOTE You only need to apply page layouts where the layout changes. Pages following a layout change use the Normal page or the parent page layout.

To create a PDF TOC for a book

1. Right click on the TOC folder in the Project Organizer and select **Add Table of Contents**.
2. Give the TOC file a name.
3. Open the TOC.
4. Drag the entire contents of the book folder from the Content Explorer to the TOC editor.
5. Rearrange the contents into the order you want them to appear in the output. The TOC for this document is illustrated below:



6. Each chapter folder will initially have a flag beside it, indicating that it is unlinked. You must link each chapter folder to the h1.chapter topic within the folder (that is, the first page of the chapter). For example, in the figure above, the highlighted Contents folder is linked to CreateContent.htm.

To link a folder:

- Right click on the folder and select Link to topic.
- Navigate to the chapter folder and select the h1 chapter topic.
- Delete the h1 chapter topic from the TOC.

7. Set up the document pagination and numbering for the following folders and topics. Right click on the folder/topic, select Properties, and use the Properties dialog to set the folder/topic properties as indicated below:

Folder/topic	Printed Output	Auto-numbers
Cover	Page number: Start at 1 Break type: Chapter Page layout: Frontmatter. For watermarked PDFs, use Watermk/Draft/Frontmatter or Watermk/Confid/Frontmatter Page type: Title Auto-end left: Disabled	Defaults
Frontmatter (Document Information)	Break type: Page Page layout: Frontmatter. For watermarked PDFs, use Watermk/Draft/Frontmatter or Watermk/Confid/Frontmatter Page type: Normal Auto-end left: Disabled	Defaults
Contents	Break type: Chapter Page layout: Chapters Page type: First Auto-end left: Disabled	Defaults
About (preface) folder	Break type: Chapter Page layout: Chapters. For watermarked PDFs, use Watermk/Draft/Chapters or Watermk/Confid/Chapters Page type: First Auto-end left: Disabled	Defaults
Chapter and appendix folders, Contents, Glossary, Index	Break type: Chapters Page layout: Chapters. For watermarked PDFs, use Watermk/Draft/Chapters or Watermk/Confid/Chapters Page type: First Auto-end left: Disabled	Reset to 1 for first chapter in book.

8. The PDF TOC contains links to the Cover and printed table of contents, which are not required when generating web output. If you are re-using the TOC for your web outputs, you need to make the Cover and

Contents conditional, so that they are included in the print outputs only. In the example above, items conditioned PrintOnly are indicated by the green box beside the TOC entry.

To create a PDF TOC for a note

1. Right click on the TOC folder in the Project Organizer and select **Add Table of Contents**.
2. Give the TOC file a name.
3. Open the TOC.
4. Drag the entire contents of the note folder from the Content Explorer to the TOC editor.
5. If your note contains more than one topic, rearrange the contents into the order you want them to appear in the output.
6. Set up the document pagination and numbering for the following folders and topics. Right click on the folder/topic, select Properties, and use the Properties dialog to set the folder/topic properties as indicated below:

Folder/topic	Printed Output	Auto-numbers
First topic	Page number: Start at 1 Break type: Page layout Page layout: Note. For watermarked PDFs, use Watermk/Draft/Note or Watermk/Confid/Note Page type: First Auto-end left: Disabled	Defaults

HTML TOCs

You can build your HTML TOCs from scratch, or by reusing PDF book TOCs, as described in ["Creating HTML TOCs" on page 72](#). You can also include other file types in the TOC, such as PDF files and videos, simply by dragging them into the TOC from the Contents folder. How you build your HTML TOCs depends on the whether you are using the HTML_FRAMES or HTML_FRAMELESS skin.

If you are using the HTML_FRAMELESS skin with the FRAMELESS_MENU master page (that includes the FRAMELESS_MENU skin component) you cannot re-use TOCs due to the way the topic menu proxy behaves (see ["HTML TOC Structure for Frameless Outputs That Use the Topic Menu Proxy" on the next page](#)).

HTML TOC Structure Differs for Frames Versus Frameless Outputs

If you are using the frameless (top-navigation) skin, you'll need to structure your TOC a bit differently than you would for Frames output, since the TOC drives the top-navigation menus. Typically this means conditioning the preface for PrintOnly, while still making it available by setting **Preface/preface.htm** as your startup topic in the target. The following table shows frameless and frames TOCs for the same book. The frameless TOC also includes a link to the PDF version of the document.

The colors indicate conditions: green is PrintOnly (set to Exclude in your HTML targets), blue is ScreenOnly (set to Include in your HTML targets), Note that although excluded topics do not show up in the output table of contents (frames) or top-navigation menus (frameless), they are still included in the output contents, and

therefore any cross-references or hyperlinks to the excluded topics work. This allows you to exclude the preface (About This Document), the Document Information and List of Styles appendix from the top-navigation menus, and link to them in alternate ways. In this case, About this Document (Preface/preface.html) is set as the Startup Topic in the target, and is displayed when the application launches, or when the user clicks on the **Home** breadcrumbs link.

The frameless TOC also includes an entry for the PDF version of the document, created by maintaining the PDF in the Contents/PDF folder and dragging that PDF into the TOC when you create the TOC. It is conditioned ScreenOnly.



NOTE If you do not condition the PDF topic ScreenOnly, and then use the same TOC to create the PDF version of the document, you will end up with the old PDF appended to the new PDF that you just generated. If you don't notice, the file will keep growing each time you generate a new version.

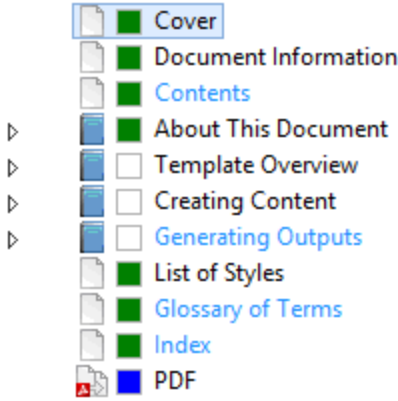
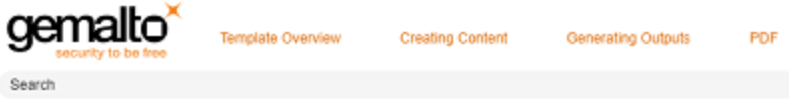
HTML TOC Structure for Frameless Outputs That Use the Topic Menu Proxy

You can add a context-sensitive topic menu to the HTML_FRAMELESS skin by formatting your topics with the FRAMELESS_MENU master page (that includes the FRAMELESS_MENU skin component)

Examples

Refer to the Flare Template project to see more example TOCs.

Frames (tri-pane) TOC	Resulting output

Frameless (top-naviagion) TOC	Resulting Output
	

Creating HTML TOCs

You need to create a single TOC for all of the books in your project. There are two ways you can do this:

- > If the project is HTML only, see ["To create an HTML TOC from scratch" below](#).
- > If the project combines HTML and PDF into a single output, see ["To create an HTML TOC by re-using the PDF book TOCs" on the next page](#).

To create an HTML TOC from scratch

Use this method if you are creating a new HTML-only TOC.

1. Right click on the TOC folder in the Project Organizer and select **Add Table of Contents**.
2. Give the TOC file a name.
3. Click Add to add the TOC to the TOC folder.
4. Open the TOC. It will have a couple of default topics. Delete them.
5. Drag the entire contents of the book folder from the Content Explorer to the TOC editor.
6. Rearrange the contents into the order you want them to appear in the output. The **Home** file should be first, followed by each individual book. Place technical notes and other content as desired.
7. Each chapter folder will initially have a flag beside it, indicating that it is unlinked. You must link each chapter folder to the h1 chapter topic within the folder (that is, the first page of the chapter). For example, in the figure above, the highlighted Contents folder is linked to **CreateContent.htm**.

To link a folder:

- a. Right click on the folder and select **Link to topic**.
 - b. Navigate to the chapter folder and select the h1.chapter topic.
 - c. Delete the h1.chapter topic from the TOC.
8. Each book folder will initially have a flag beside it, indicating that it is unlinked. You must link each book folder to the **preface.htm** file in the **Preface** folder for the document (the first page of the preface, that briefly describes the document and provides links to each chapter in the document).

To create an HTML TOC by re-using the PDF book TOCs

Use this method if you are creating an HTML TOC that contains documents for which TOCs already exist. For example, if you are creating a TOC that contains multiple books for which individual TOCs already exist.



NOTE Do not use this method if you are using the HTML_FRAMELESS skin with the Menu proxy. See ["Limitations When Re-using TOCs" on page 66.](#)

1. Right click on the TOC folder in the Project Organizer and select **Add Table of Contents**.
2. Give the TOC file a name.
3. Click Add to add the TOC to the TOC folder.
4. Open the TOC. It will have a couple of default topics. Delete them.
5. Drag the individual TOCs you want to re-use from the **Project/TOCs** folder to the TOC editor.
6. Rearrange the contents into the order you want them to appear in the output. The **Home** file should be first, followed by each individual book. Place technical notes and other content as desired.



NOTE Some topics in the TOC, such as the cover or the printed table of contents may be applicable to only one output. You can use conditional text to control which topic to include on the printed and online outputs when re-using TOCs.

Configuring Your Output Targets

After you have created your content and TOCs, you need to configure targets that specify how you want to publish your project. You must create a target for each individual output. In a multi-book project, you would create one target for each PDF book, and one target for the HTML output. Targets allow you to specify the following attributes for your project outputs:

- > the format of the output. This template currently supports PDF and HTML. Other formats, such as MSWord and EPUB are available.
- > the TOC used to build the output.
- > the conditional text tags you want to include or exclude in the output.
- > the variable definitions, such as document title, that you want to use in the output.
- > where to generate and, optionally, publish the output.
- > other options which are not used in this template.

Batch targets

You can group two or more targets together to create a batch target that builds all of your outputs at once. This template uses batch targets to create an integrated set of outputs that contain all of the documentation for the project in HTML and PDF formats. See ["Creating Batch Targets" on page 78.](#)

Creating the PDF Targets

You must create a target for each PDF output in your project. If your project contains multiple books, configure each PDF target to publish the PDF book to a PDF folder outside of the project folder. These PDF files are then copied to the PDF folder inside the project. This allows you to create a link to each PDF book from the Home (or splash) page.



NOTE You must publish the PDF files to a folder outside of the project since attempting to publish directly to the PDF folder in the project's Content folder causes Flare to complain. See ["Building and Publishing Output Targets" on page 80](#) for more information.

To create a PDF target

1. Right-click on the **Targets** folder in the Project Organizer and select **Add Target**.
2. Give the target a name and specify **PDF** as the output type. **Click Add**.
3. Open the target in the Target Editor.
4. Click the **General** tab and configure the general attributes as follows:

Output Type	PDF
Comment	Optional
Master TOC	The TOC for the book.
Master Page Layout	(default)
Master Stylesheet	/Resources/Stylesheets/Gemalto_Template
Output File	(default)
Output Folder	(default)

5. Click the **Conditional Text** tab and configure the conditional text attributes as follows:

Default.Note	Exclude
Default.PrintOnly	Include
Default.ScreenOnly	Exclude

If you are using other condition tags, configure them as required.

6. Click on the **Variables** tab and enter target-specific overrides for the default variable definitions as required.
7. Click on the **Publishing** tab to specify the destination where you want to publish the PDF. If the PDF destination already exists, click the appropriate Publish checkbox. If no PDF destination is listed, create it as follows:
 - a. Click **New Destination**.

- b. Select **New From Template**, enter **PDF** in the **Filename** field, and click **Add**.
- c. Configure the destination as follows:

Type	File System. This is the simplest destination type, and allows you to publish to your local computer, or to a mounted drive. You can create other destination types, (for example FTP or SFTP) that allow you to publish to remote servers using login credentials. See the Flare documentation for more information.
Comment	As desired
Directory	For single-source projects that combine HTML and PDF outputs, you need to copy the PDF files to the PDF folder in the project's Content folder. See "Building and Publishing Output Targets" on page 80 for details. If doing a local build using the Flare UI, you will be warned that you are publishing to the Contents folder. If you are using madbuild to automate you build process, then you must publish to a location outside of the project and then copy the PDFs to the Contents/PDF folder as part of the script. This is because the warning stops madbuild and waits for confirmation, but you are not there to give it, and then it really isn't automated, is it?
Publish Options	Uncheck the Upload Only Changed Files and Remove Stale Files checkboxes.

- d. Save the destination file. You can use it for all of your other PDF targets.
8. Click on the **Advanced** tab and configure the options as follows:
 - a. Uncheck the **Generate TOC proxy**, **Generate index proxy** and **Generate glossary proxy** checkboxes. Otherwise Flare will insert a bogus TOC, index, and glossary into your document, and you will wonder where that came from.
 - b. Set the **Stylesheet Medium** to **Print**.
 9. Click on the PDF Options tab and set the PDF options as follows:
 - a. Enter the document title in the **Title** field of the Document Properties section.
 - b. Enter **Gemalto** in the **Author** field of the Document Properties section.
 - c. Uncheck **Include non-TOC bookmarks in the bookmarks pane**.
 - d. Set the Initial View options as follows:
 - i. Set **Magnification** to **Fit Page**.
 - ii. Set Navigation to **Bookmarks Panel and Page**.
 - iii. Set **Page Layout** to **(default)**.
 - iv. Set **Title bar** to **Document Title**.
 - v. Check the **Collapsed bookmarks** checkbox.
 10. Set any other options as required. Refer to the Flare documentation for details.

Creating the HTML Target

You must create a target for the HTML output.

To create an HTML target

1. Right-click on the **Targets** folder in the Project Organizer and select **Add Target**.
2. Give the target a name and specify HTML5 as the output type. Click **Add**.




NOTE By convention, HTML targets are prefaced with XHTML_ to ensure that they appear at the end of the batch target. This helps to ensure that the HTML target is built last in the batch target, with alphabetical order being the default, although you may have to edit a batch file target to define the correct order, as described in ["Creating Batch Targets" on page 78](#).

3. Open the target in the Target Editor.
4. Click the **General** tab and configure the general attributes as follows:

Output Type	HTML5
Comment	Optional
Startup Topic	If set to (default), the first file in the TOC is displayed. For multi-document projects, set to /Content/Home.htm . For single-document projects set to /Content/<document>/Preface/Preface.htm .
Master TOC	Specify the TOC for the target.
Browse Sequence	(default)
Master Stylesheet	/Resources/Stylesheets/Gemalto_Template
Output File	Use the default (Default.htm) or index.html.
Output Folder	(default)

5. Click on the **Skin** tab and configure the skin attributes as follows:

General	<p>Select the HTML_FRAMES or HTML_FRAMELESS skin as desired.</p> <div>  <p>NOTE If you select the HTML_FRAMELESS skin, you can add a Topic Toolbar proxy (for print, remove highlight, next, previous buttons) and/or a Menu proxy (for context-sensitive side navigation) by specifying different master pages, as described in step 10. See also HTML Top-Navigation Outputs for more information.</p> </div>
Component Default Skins	(default)

Responsive Output Settings	Default break points (width at which the menus are replaced by the "hamburger" icon) are defined in the skin. You can override them here as required to match the design of your project. Typically, you want to set the break point such that "hamburger" icon appears before the top-level menus wrap to two lines.
-----------------------------------	---

6. Click the **Conditional Text** tab and configure the conditional text attributes as follows:

Default.Note	Exclude
Default.PrintOnly	Exclude
Default.ScreenOnly	Include

If you are using other condition tags, configure them as required.

7. Click on the **Variables** tab and enter target-specific overrides for the default variable definitions as required. See ["Using Variables" on page 52](#) for more information.
8. Click on the **Publishing** tab to configure the target to publish the HTML content, including the Home page and the linked PDF files to the publish destination for the finished project. This could be a web server on which you want to publish the output or a file server that development will use to get the documentation for loadbuild, for example. You can create as many publish destinations as you require. If necessary, you can create a destination for the HTML output as follows:
- Click **New Destination**.
 - Select **New From Template**, enter **XHTML_All_Docs** in the **Filename** field, and click **Add**.
 - Configure the destination as follows:

Type	As required.
Comment	Enter "Publish entire project to web", for example.
Directory	As required.
Publish Options	Uncheck the Upload Only Changed Files checkbox. Check the Remove Stale Files checkbox.

- Save the destination file.
9. Use the **Glossary** tab to specify the glossary you want to include in your output, if relevant.
10. Click on the **Advanced** tab and set the options as follows:
- Check the **Exclude content not linked directly or indirectly from the target** checkbox. If you do not, everything in the Content folder is included in the output and, most importantly, the search. This may lead to erroneous search results.
 - Check the **Add meta tags to content** checkbox to add the following meta tag to the output. This tag makes IE work properly:

```
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
```
 - Set the Master Page as follows:

Default	For standard tri-pane outputs
FRAMELESS	For basic top-navigation output with breadcrumbs only
FRAMELESS_TOOLBAR	For top-navigation output with breadcrumbs and toolbar. See Configuring your outputs to use the HTML_FRAMELESS skin with the FRAMELESS_TOOLBAR master page for additional configuration information.
FRAMELESS_MENU	<p>For top-navigation output with breadcrumbs, toolbar, and context-sensitive topic menu. You must configure the master page to specify the TOC to use to populate the context-sensitive topic menu, as follows:</p> <ol style="list-style-type: none"> 1. Open the FRAMELESS_MENU master page (Content/Resources/MasterPages/FRAMELESS_MENU.flmsp) 2. Right-click on the Menu Proxy block and select Edit Menu Proxy. 3. Use the Use TOC, Browse Sequence, or Headings dropdown menu to select the TOC to use to populate the menu. This should be your project's TOC. 4. Set the other options as follows: <ol style="list-style-type: none"> a. Check all of the checkboxes so that your menu is context sensitive and includes parent, children, and sibling topics. b. Set the number of levels to display to 3. You can adjust this setting if required. c. Set the skin file to Project/Skins/HTML_FRAMELESS <p>See Configuring your outputs to use the HTML_FRAMELESS skin with the FRAMELESS_MENU master page for additional configuration information.</p>

- d. Set the **Stylesheet Medium** to **non-print**.
- e. Click on the **Performance** tab to specify various options that may affect build and viewing performance. You can accept the defaults. In the **Search Database** section, you may want to check the **Enable Partial Word Searching** checkbox to have the search function find matches for 3-letter strings embedded in a word. This is useful if you are doing API documentation or other types of documentation where someone may use a partial string (for example AES) to search for all terms containing that string.



TIP Enabling partial word searching dramatically increases the size of the search database and build times. If you use this feature, you may want to enable it only when building your final outputs for configuration management.

5. Set any other options as required. Refer to the Flare documentation for details.

Creating Batch Targets

Batch targets list all of the targets in the project. Checkboxes allow you to select which individual targets you want to build and/or publish as part of a batch job. When executed, Flare builds and publishes all of the selected targets **in the order specified in the batch target**.

A batch target is used in this template to do the following:

1. Build each PDF output and publish it to the **Contents/PDF** folder.

2. Build the HTML output and publish it to the publish destination specified in the HTML target.

Build and publish the PDFs before building and publishing the HTML

If you are using a batch target to build a document set that includes HTML and PDF, you must build and publish the PDFs first, so that when the batch file builds and publishes the HTML, the **Contents/PDF** folder contains up-to-date PDFs. To ensure that the PDFs are built and published before the HTML, you can take several different approaches:

1. Name your HTML targets so that they always appear at the end of the batch file. By convention, HTML targets are named XHTML_<docset> to achieve this.



CAUTION! This method only works if the target names are alphabetically correct when you create the batch target. If you change the name of an HTML target after the batch target is created, it will appear to be at the end of the file in the Flare UI, but in the actual file (<batch_target>.flbat) it will still be in the same position. To ensure that the HTML file appears last, you can manually edit the batch file (using a text editor) to ensure that the HTML target is built last, or delete the batch file and create a new one.

2. Split your build in two steps, so that the PDF batch runs first, followed by the HTML build.
3. Publish the PDFs to a location outside of the project, and then use the Windows **xcopy** command in a build script to copy the PDFs to the Content/PDF folder in the published HTML output.

See ["Automated Building and Publishing" on page 81](#) for more information and example build scripts.

To create a batch target

1. Right-click on the Targets folder in the Project Organizer and select **Add Target**.
2. Give the target a name and specify **Batch Target** as the file type. Click **Add**.
3. Open the target in the Batch Target Editor. All of the targets contained in the **Targets** folder in the Project Organizer are listed.
4. Click the **Build** and **Publish** check boxes for each PDF output and for the HTML output.
5. Save the batch target. It is saved to the **Targets** folder in the Project Organizer.

Setting Up Your Project for Combined HTML and PDF Outputs

This template supports integrated HTML and PDF outputs where the HTML and PDF contain identical content, providing users the choice to view the documentation in HTML or PDF format. In this set up, you include PDF versions of the books in the project in the project's Content folder so that you can create links from the HTML home page to the individual PDF books, as described in ["Creating the Home Page" on page 57](#).

To do this, you create a PDF folder in the Content directory, add the PDF files for the individual books to the folder, and then use the **Insert > Hyperlinks** function to create links from the HTML to the PDF files. This is a one-time task. Once the links are created, you can overwrite the PDF files at build time with updated files that have the same file name.



CAUTION! When building a project that combines HTML and PDF outputs, the PDFs must be built and published before the HTML is built and published, so that the HTML includes up-to-date PDFs when it is published. If you are building and publishing manually, simply build and publish your PDFs first. If you are building automatically, there are several methods to ensure the PDFs are up-to-date in the published HTML, as detailed in ["Creating Batch Targets" on page 78](#).

To set up your project for combined HTML and PDF outputs

1. Create a PDF folder in the project's Contents directory.
2. Create a Destination for the PDF and set the path to the project's Contents/PDF directory.
3. Set the publishing destination in the target (Publishing tab) to the destination you just created.
4. Create a batch target for all of your PDF files and set each file to build and publish.
5. Build the batch target to create the PDF and publish them to the Content/PDF folder.
6. Create hyperlinks from the HTML Home page to the PDF files in the Content/PDF folder.

Once you have set up the folders and hyperlinks, you simply need to build the batch target first, followed by the HTML target to create the combined HTML/PDF project. Ensure that the file names do not change. All of the hyperlinks will remain. Although you can do this manually, this is best achieved using a build script, as described in ["Building and Publishing Output Targets" below](#).

Building and Publishing Output Targets

After you have created targets for all of your outputs, you can use the targets to build and publish your outputs:

- > When you **build** an output, the output is saved to the **Output**/**<user>** folder located in the same directory as your project file. If you are simply emailing files around for reviews and configuration management, this is adequate, and you do not need to publish your outputs.
- > When you **publish** an output, the output is copied from the **Output** folder to the location specified by the Destination associated with the target.

Outputs that combine HTML and PDF

If your project combines HTML and PDF targets into a single package that provides links from the Home page or TOC to PDF versions of the documentation, the PDF files must exist, and be updated in, the **Content** folder, so that you can link them. See ["Setting Up Your Project for Combined HTML and PDF Outputs" on the previous page](#) for more information.

By convention, the PDF files are placed in the **Content/PDF** folder. You can publish your PDFs directly to the **Contents/PDF** folder, but you must ensure that you publish your PDFs before you publish the HTML to ensure that the HTML output includes up-to-date PDFs.

Manual Building and Publishing

You can manually build or publish an output by right-clicking on the output target and selecting either **Build** or **Publish**. Batch targets allow you to build and publish in a single step. Note that you must build an output target before you can publish it. Manual building and publishing is useful for small projects, or if you just want to check your work. For large projects, it is recommended that you automate your building and publishing tasks to save time and ensure consistency and thoroughness.

Automated Building and Publishing

There are two ways you can automate the building and publishing of your output targets:

- > using Scheduled Tasks in Windows.
- > using the **madbuild** command line tool in a Bash script

Scheduled Tasks

You can set up scheduled tasks in windows to automate your output target building and publishing. Refer to the Flare and Windows documentation for more information.

Using the madbuild Command

The **madbuild** command allows you to build a target or batch target from the command line. You can run madbuild from a command prompt window, or you can call it from a Bash script. Bash scripts also allow you to automate other parts of the publishing process, such as integrating non-Flare documentation within your Flare project, or publishing to a web site.

Batch (BAT) Scripts

Batch files provide the greatest flexibility for building and publishing your output targets. You can run batch commands on demand, or as a scheduled task. You could further enhance the build process by creating interactive scripts that prompt for information such as where to publish. Another potential enhancement is a log parser that reports build errors found in the logs.

Sample BAT script that uses madbuild to build a PDF batch target followed by an HTML target

The following batch file is used to build the Enterprise template outputs in two steps. It builds and publishes the PDFs first, followed by the HTML:

Figure 8: Sample BAT script used to build the Enterprise template project

```
@echo off
REM Set global variables
set FLARE="C:\Program Files\MadCap Software\MadCap Flare 13\Flare.app"
set PROJECT="C:\CustomerDoc\Templates\2017\Enterprise\Gemalto_Flare_Template.flprj"
REM Set target variables
set TARGET_HTML="XHTML_ALL_DOCS.fltar"
set TARGET_PDF_BATCH="PDF\BATCH_PDF.flbat"
REM Build the project
cd %FLARE%
REM Build and publish the targets
REM Always build the PDF targets first
REM so that the HTML builds include up-to-date PDFs
madbuild -project %PROJECT% -batch %TARGET_PDF_BATCH% -log true
madbuild -project %PROJECT% -batch %TARGET_HTML% -log true
REM cmd \k
```

This script builds and publishes the outputs for a combined HTML/PDF project in two steps, by building and publishing the PDFs first, and the HTML second. The script works as follows:

1. Set the variables for the script. Using variables is recommended since it allows you to easily update and re-use the script. The variables are defined as follows:

FLARE	Specifies the full path to the Flare executable.
PROJECT	Specifies the full path to the Flare project you want to build.
TARGET_HTML	Specifies the path, relative to the Project/Targets folder, to the HTML target you want to build.
TARGET_PDF_BATCH	Specifies the path, relative to the Project/Targets folder, to the PDF batch target you want to build.

2. Build the projects using **madbuild**. To run **madbuild**, you must first **cd** to the folder containing the **madbuild** executable (**cd %FLARE%**).

The **madbuild** executable takes as arguments the project and target you want to build.

- If you are building a single target, the syntax is **madbuild -project <project> -target <target> [-log true]**
- If you are building a batch target, the syntax is **madbuild -project <project> -batch <batch_target> [-log true]**

Sample BAT script that uses madbuild to build a PDF/HTML batch target and then copy the PDFs to the published HTML output

The following batch file is used to build the Enterprise template outputs using a single batch target. The PDFs are published to a location outside of the project and then copied to the published HTML output after the build is complete:

Figure 9: Sample BAT script used to build the Enterprise template project

```
@echo off
REM Set global variables
set FLARE="C:\Program Files\MadCap Software\MadCap Flare 13\Flare.app"
set PROJECT="C:\CustomerDoc\Templates\2017\Enterprise\flare\Gemalto_Flare_Template.flprj"
REM Set target variables
set TARGET_HTML_PDF_BATCH="HTML_PDF_BATCH.flbat"
REM Set PDF variables
set PDF_SRC="C:\CustomerDoc\Templates\2017\Enterprise\flare\pdf"
set PDF_DEST="C:\inetpub\wwwroot\Templates\2017\Enterprise\Content\PDF"
REM Build the project
cd %FLARE%
REM Build and publish the targets
REM Always build the PDF targets first
REM so that the HTML builds include up-to-date PDFs
madbuild -project %PROJECT% -batch %TARGET_HTML_PDF_BATCH% -log true
REM Copy the built PDF to the output destination
xcopy /s /y %PDF_SRC% %PDF_DEST%
REM cmd \k
```

The script works as follows:

1. Set the variables for the script. Using variables is recommended since it allows you to easily update and re-use the script. The variables are defined as follows:

FLARE	Specifies the full path to the Flare executable.
PROJECT	Specifies the full path to the Flare project you want to build.
TARGET_HTML_PDF_BATCH	Specifies the path, relative to the Project/Targets folder, to the PDF batch target you want to build.
PDF_SRC	Specifies the location where the PDFs were published using the build script.
PDF_DEST	Specifies the location on the built HTML output to copy the updated PDFs to.

2. Build the projects using **madbuild**. To run **madbuild**, you must first **cd** to the folder containing the **madbuild** executable (**cd %FLARE%**).

The **madbuild** executable takes as arguments the project and target you want to build.

- If you are building a single target, the syntax is **madbuild -project <project> -target <target> [-log true]**
- If you are building a batch target, the syntax is **madbuild -project <project> -batch <batch_target> [-log true]**

3. Copy the PDFs from the PDF_SRC folder to the PDF_DEST folder.

Publishing to a Web Server

Publishing your documentation to an internal web server is strongly recommended. By publishing to a web server, your documentation is easily accessible to your internal development, verification, and support teams, amongst others. To maximize the benefits of publishing to a web server, it is recommended that you publish your documentation on a regular basis, preferably using an automated nightly build, with an on-demand option.

To publish to a web server, the best approach is to use a single server (for example, Windows Server 2008 or Windows Server 2012) configured as both a file server and web server. This server is used to host the documentation source (file server) and generated targets (web server). Using this method, you can run the build scripts as Administrator to locally access the source and locally publish the outputs to the web server (which requires Administrator access privileges), avoiding any network and permissions issues.

The example build script shown in ["Automated Building and Publishing" on page 81](#) illustrates this concept. In this setup, the source resides on the server in the **C:\CustomerDoc** folder. The web server content resides under **C:\inetpub\wwwroot** in product-specific folders, which in turn contain release-specific folders, as illustrated below:

Figure 10: Web server folder structure

```
C:\inetpub\wwwroot
  Product A
  Product B
    1.0
    2.0
```

APPENDIX A: List of Styles

This appendix lists the styles used in this template with a brief description and an example of the style format.

Generic Classes

You can apply a generic class to any paragraph or heading style. For example, to insert a page break before a paragraph, select the paragraph and then use the Styles drop-down to apply the **.pagebreak** generic class.

.ntoc

Use this class to exclude an h1, h2, or h3 from the table of contents.

.pagebreak

Use this class to insert a page break above the selected paragraph.

Heading Styles

Heading styles are used to start new chapters or topics, and to arrange the content of your topics into concepts, tasks, and procedures.

h1

Typically not used in books, but can be used to break up technical notes or release notes into major sections. In print outputs, automatically inserts a page break.

h1.appendix

Use this class for appendices. For print outputs, adds the autonumber A:APPENDIX {A+}. The autonumber displays as APPENDIX n: in the heading, and as Appendix n: in the header.

h1.chapter

Use this class for chapters. For print outputs, adds the autonumber CHAPTER {chapnum}. The autonumber displays as CHAPTER n: in the heading, and as Chapter n: in the header.

h1.contents

Use this class for the print table of contents. Not used for online outputs. For print outputs, adds the autonumber CONTENTS. The autonumber displays as CONTENTS in the heading, and as Contents in the header.

h1.index

Use this class for the print index. Not used for online outputs. For print outputs, adds the autonumber INDEX. The autonumber displays as INDEX in the heading, and as Index in the header.

h1.preface

Use this class for the preface. For print outputs, adds the autonumber PREFACE. The autonumber displays as PREFACE in the heading, and as Preface in the header. Recommended preface heading is "About the <title>".

h2

Used for introduce individual topics within a chapter. It defines the default style characteristics for all of the h2 classes. Each h2 section is a separate HTML file.

h2.ntoc

Used for h2-level headings that you do not want to appear in the printed table of contents.

h2.pageBreak

Used for h2-level headings that you want to start on a new page in the print output. An example would be a command in a command reference guide.

h3

Used for level-three headings. h3 headings are nested within a h2 topic file. Do not use h3 headings to start an individual topic file.

h3.ntoc

Used for h3-level headings that you do not want to appear in the printed table of contents.

h4

Used for level-four headings. Also used for procedure headings so that all procedures look the same.

This is an example h4 heading

h5

Used for procedure headings.

This is an example h5 heading

Paragraph Styles

p

Used for paragraphs. This is an example p style.

p.caution

Used for cautions, as shown below. Automatically adds the graphic and CAUTION: autonumber.



CAUTION! Exercise caution. Contains important information that may help prevent unexpected results or data loss.

p.figureCaption

Used for figure captions. Inserts the autonumber F:Figure {n+}:

Figure 11: This is an example figure caption

p.footer

Used for the first line of the footer (document title) in the print output.

This is an example of the p.footer style

p.footerCopyright

Used for the second line of the footer (copyright, etc.) in the print output.

This is an example of the p.footerCopyright style

p.header

Used for the headers in the print output. The header content is auto-populated.

This is an example of the p.header style

p.home

Used for the breadcrumbs **Home** link in the online output.

This is an example of the p.home style

p.listLevel1

Used to insert a paragraph within a level1 list.

This is an example of the p.listLevel1 style

p.listLevel2

Used to insert a paragraph within a level2 list.

This is an example of the p.listLevel1 style

p.listLevel3

Used to insert a paragraph within a level3 list.

This is an example of the p.listLevel1 style

p.note

Used for notes, as shown below. Automatically adds the graphic and Note: autonumber.



NOTE Exercise caution. Contains important information that may help prevent unexpected results or data loss.

p.ol1

Used for level1 ordered list items.

1. This is an example of the p.ol1 style

p.ol1Start

Used for the first item in a level1 ordered list. Resets the counter to 1.

1. This is an example of the p.ol1Start style

p.ol2

Used for level2 ordered list items.

a. This is an example of the p.ol2 style

p.ol2Start

Used for the first item in a level2 ordered list. Resets the counter to a.

a. This is an example of the p.ol2Start style

p.ol3

Used for level3 ordered list items.

i. This is an example of the p.ol3 style

p.ol3Start

Used for the first item in a level3 ordered list. Resets the counter to i.

i. This is an example of the p.ol3Start style

p.pagenum

Used to display the page number in the footer. Uses PageNumber variable.

This is an example of the p.pagenum style

p.productLine

Used on the book cover and footer. Populated by the <Product> variable.

This is an example of the p.productLine style

p.productLineNotes

Deprecated. Use p.productLine instead (formatting is identical).

This is an example of the p.productLineNotes style

p.subjectNotes

Used for the subject heading in technical notes. Populated by the <SubjectNotes> variable.

This is an example of the p.subjectNotes style

p.tip

Used for tips, as shown below. Automatically adds the graphic and TIP autonumber.



TIP This is some information that will allow you to complete your task more efficiently.

p.title

Used on the book cover and footer. Populated by the <Title> variable.

THIS IS AN EXAMPLE OF THE P.TITLE STYLE

p.titleNotes

Deprecated. Use p.title instead (formatting is identical).

THIS IS AN EXAMPLE OF THE P.TITLENOTES STYLE

p.ul1

Used for level1 unordered list items.

> This is an example of the p.ul1 style

p.ul2

Used for level2 unordered list items.

- This is an example of the p.ul2 style

p.ul3

Used for level3 unordered list items.

- This is an example of the p.ul3 style

p.warning

Used for warnings, as shown below. Automatically adds the graphic and WARNING! autonumber.



****WARNING**** Exercise caution. Contains important information that may help prevent unexpected results or data loss.

pre

Used for preformatted text, such as command output examples, so that the formatting is retained.

This is an example of the pre style

pre.listlevel1

Used to insert a preformatted text within a level1 list.

This is an example of the pre.listLevel1 style

pre.listlevel2

Used to insert a preformatted text within a level2 list.

This is an example of the pre.listLevel2 style

pre.listlevel3

Used to insert a preformatted text within a level3 list.

This is an example of the pre.listLevel3 style

APPENDIX B: Glossary of Terms

A

A term

A definition

Ab term

Ab definition

Ac term

Ac definition

B

B term

B definition

C

C term

C definition

D

D term

D definition

E

E term

E definition

F

F term

F definition

G

G term

G definiton

H

H term

H definition

I

I term

I definition

J

J term

J definition

K

K term

K definition

L

L term

L definition

M

M term

M definition

N

N term

N definition

O

O term

O definition

P

P term

P definition

Q

Q term

Q definition

R

R term

R definition

S

S term

S definition

T

T term

T definition

U

U term

U definition

V

V term

V definition

W

W term

W definition

X

X term

X definition

Y

Y term

Y definition

Z

Z term

Z definition

INDEX

DRAFT