FM2DITA User Guide

v.1.05
2020-04-06

Leximation, Inc.

FM2DITA User Guide.
by Scott Prentice, Leximation, Inc.

Published by Leximation, Inc., 122 H Street, San Rafael, CA 94901.

This document was authored and published using FrameMaker and DITA-FMx.

The most current version of this guide is available on the Internet at
http://docs.leximation.com/fm2dita/1.05/

Contents

1 Using FM2DITA

FM2DITA is a FrameMaker plugin that provides numerous tools and utilities that assist in the process of converting unstructured FrameMaker files to DITA XML.

This plugin supports FrameMaker versions 7.2, 8, 9, 10, 11, 12, 2015, 2017, and 2019.

Documentation updated on April 6, 2020 for version 1.05.

FM2DITA will continue to be updated over time. Check for updates to get the latest tools!

FM2DITA provides the following types of automation:

- Tools for analyzing existing content such as counting topics in a book, as well as generating reports of style, object, and conditional text usage
- Preconversion tools for cleaning up, retagging, and renaming objects in the unstructured FM files
- Processing of structured FM binary files to assist with cleanup and refactoring into proper DITA structures (see the list of commands for details)
- Breaks each FM file into multiple DITA XML topics, and generates root and chapter maps preserving the hierarchy that exists in the unstructured files

Contact us at <tools AT leximation DOT com>.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Requirements" on page 2
"Editing the fm2dita.ini File" on page 26
"Programmatic Control of FM2DITA" on page 46

Requirements

Assumptions and requirements for effective use of FM2DITA.

FM2DITA is a plugin that provides commands and utilities that assist with the process of converting unstructured FrameMaker documents into DITA. It is not a magic bullet that does everything for you (if that's what you're looking for, hire a consultant). If used properly, it will allow you to establish a reliable and consistent conversion process and will save you a significant amount of time.

The following list describes the additional tools (both software and mental) that will likely be required to make effective use of this plugin.

- *FrameMaker.* Version 7.2 or later.
- *DITA support in FrameMaker.* We recommend that you use DITA-FMx to provide DITA support in FrameMaker (all versions), but you can use this plugin with the default DITA support as well.
- *Conversion table development expertise.* This plugin assumes that you have developed a conversion table that does the initial conversion from unstructured FM files to structured FM files. FM2DITA does provide "preconversion" tools and utilities that you may want to use before applying the conversion table, but it does not create the conversion table for you. If you do not have conversion table development expertise, you can have this created for you by a consultant, and you'll be able to use this plugin to perform the remainder of the conversion yourself.
- *DITA Open Toolkit or other validation tools.* After completing the conversion process and breaking the structured FM files into DITA topics and maps, it is often useful to have a command line tool that you can run to do a quick sanity check validation on the resulting files. This is not required, but will help to find any problems that might become evident later.

RELATED INFORMATION:
"Installation and Setup" on page 3
"Limitations of an FM2DITA Trial" on page 2

Limitations of an FM2DITA Trial

The trial version of FM2DITA imposes certain limitations on usage.

These limitations should be expected when using the FM2DITA trial.

- No more than three documents in a book will be processed when using the "book" commands. For efficient testing, it's best to delete all but three files from your book.
- Each command has a limit on the number of operations that can be performed. The actual number of operations varies for each command, hopefully providing enough functionality to show the usefulness of each command. Some of the limits are a total aggregate limit and others will be per document.

A fully licensed version of FM2DITA imposes no restrictions or limitations on the number of documents or operations.

RELATED INFORMATION:
"Installation and Setup" on page 3
"Sample Files" on page 5

Installation and Setup

FM2DITA is easy to install and use.

RELATED INFORMATION:
"Requirements" on page 2
"Limitations of an FM2DITA Trial" on page 2
"Install the Structure Application" on page 4
"Sample Files" on page 5
"Uninstalling FM2DITA" on page 6

Run the Installer

The installer application copies the program and support files into the file system.

During the installation process the *maker.ini* file is modified, if you'd like to preserve a copy of this file in its current state, make a backup before running the installer.

1) Extract the installer executable file from the ZIP file.

2) Run the installer and select the FrameMaker version to install into.

If this is the first time you are installing FM2DITA you will need to enter an authorization code before it can be used. You can request a 30-day trial code by choosing the **Pubs-Tools > FM2DITA > Try Now** command. Enter your trial

or full authorization code by choosing the **Pubs-Tools > FM2DITA > Enter Authorization Code** command

If you are reinstalling, there is no need to uninstall the old version, but you may want to make a backup of the *fm2dita.dll* file in case you want to roll back to the previous version.

RELATED INFORMATION:
"Installation and Setup" on page 3

Install the Structure Application

A DITA 1.1 structure application is provided as one possible option.

You are free to use your own structure application, but you may want to install the FM2DITA application to use for testing and review. We suggest using a DITA 1.1 structure application even if you are creating DITA 1.2 files. The FM2DITA processing tools don't currently offer support for any DITA 1.2 features (keyref, conref ranges, etc.) so there is no need to use a DITA 1.2 application.

Whatever structure application you use, it should reference and be built on the ditabase model. Assuming that you'll be generating multiple topic types from chapter FM files, you need to use an EDD model that supports multiple XML models. If you're just writing one topic type, you can use a structure application that is specific to that topic type.

NOTE: *For FrameMaker 7.2 users, menu items indicated on the* ***StructureTools*** *menu can be found on the* ***File*** *menu.*

1) Select a location where you'll store your structure applications. The default location that FrameMaker uses is in the FrameMaker program folder at *FrameMaker\Structure\xml*. You can use that location or you may want to use a location outside of the program folder structure (current versions of Windows make it difficult to edit files in the program files area). For ease of use, we suggest creating a folder called *structapps* in the *%appdata%\Adobe\FrameMaker\<ver>* folder.

2) Extract the contents of the *FM2DITA_1.1_apps.zip* file to the folder defined in step #1. This will create a folder named *FM2DITA_1.1* that contains folders named *dtd* and *Topic*. These folders contain the Topic structure application as well as the DITA 1.1 DTD files used by the applications.

3) Start FrameMaker and open the structure application definitions file (**StructureTools > Edit Application Definitions**).

4) Open the Structure View window. In the structure application definitions file place the insertion point just after the Version element. When the

insertion point is in the right location, you'll see a black triangle pointing to the right in the Structure View window.

5) Choose **File > Import > File**, then navigate to the *struct-apps-stub_topic_1.1.fm* file in the *FM2DITA_1.1\Topic* folder created earlier. Select the Import by Reference option and choose the Import button. In the next dialog accept the defaults and choose Import.

6) Save the *structapps.fm* file, then choose **StructureTools > Read Application Definitions**.

7) Close the *structapps.fm* file and exit FrameMaker.

RELATED INFORMATION:
"Installation and Setup" on page 3

Sample Files

A sample conversion table and content files are provided for testing and review.

FM2DITA installs a file named *FM2DITA_samples.zip*, which contains a structure application and sample FrameMaker content that can be used for testing and review. This ZIP file will be installed to the *FrameMaker\Pubs-Tools* folder.

Copy this file to a location on your system and extract the contents. After extraction, you'll see a folder named *fm2dita-samples* that contains three folders, *source*, *images*, and *out*. It also contains a file named *f2d-conv-table.fm* (this is the sample conversion table).

The conversion table is set up to convert the sample unstructured FM files to structured files, and it contains notes that explain each row in the table.

All of the FM files are saved as version 7.2 files, so if you're opening them on a later version of FrameMaker, you may want to open them all and save them so they are saved as the proper version to avoid needless messages.

If you're using a Trial version of FM2DITA, you'll want to remove two of the files from the book, since the Trial version doesn't allow processing of more than 3 files (you'll get a message each time you try to process a book with more than 3 files).

To run the full conversion on the sample files, you can follow the guidelines in Typical Conversion Process. In Task 3, where you run specific FM2DITA commands, you only need to use the following commands (the provided *fm2dita.ini* file is set up as needed for these commands).

- **Assign IDs to Topics in Book**
- **Fix Images in Book**

- **Fix Tables in Book**
- **Fix Cross-refs in Book**
- **Flatten Cross-refs in Book**
- **Move Markers in Book**
- **Delete Invalid Attributes in Book** - Delete the Id and Idref attributes
- **Delete Unstructured Markers in Book** - Delete Cross-Ref markers

RELATED INFORMATION:
"Installation and Setup" on page 3

Uninstalling FM2DITA

No "uninstall" application is provided.

To disable the FM2DITA plugin, you can "comment out" the FM2DITA line in the *maker.ini* file by adding a semicolon at the beginning.

```
;Pubs-Tools:FM2DITA=Standard, FM2DITA, Pubs-Tools\fm2dita.dll, structured
```

To remove the FM2DITA plugin, follow these steps:

1) Locate the Pubs-Tools:FM2DITA entry in the APIClients section of your *maker.ini* file (in the main *FrameMaker* folder).

2) Delete this line from the INI file, then save and close the file.

3) To completely remove the plugin files, delete the *FM2DITA.** files from the *Pubs-Tools* folder located in the main *FrameMaker* folder.

RELATED INFORMATION:
"Installation and Setup" on page 3

Typical Conversion Process

Each set of unstructured FrameMaker files will likely have its own specific conversion requirements. The process described here outlines the tasks in a typical conversion.

This section breaks the conversion process into general tasks, and within each task are steps that may apply to your conversion process.

Do not use this as an absolute guide. Your files may require additional processing or may not need everything that is indicated. Additionally, each of the commands provided by this plugin can be customized in many ways; be sure to read the documentation carefully to ensure that you're getting the most from the tool.

REMEMBER: *It is important to fully understand the conversion process and what each FM2DITA command is doing to your files. Blindly running commands, will likely result in problems that are hard to debug.*

When an FM2DITA command is used, it reads default settings from the *fm2dita.ini* file. The *fm2dita.ini* file is located by checking the current directory (the directory with the file or book being processed), then each parent directory. If this INI file is not found, it reads from the default file in the *Pubs-Tools* folder (use **Pubs-Tools > Open Pubs-Tools Folder** to locate this folder).

TIP: *We encourage you to purchase the FrameSLT plugin from West Street Consulting (weststreetconsulting.com). The NodeWizard feature in this plugin is invaluable for performing various types of cleanup in the structured FM files.*

RELATED INFORMATION:
"Typical Conversion Process" on page 6
"Task 1: Preconversion Cleanup" on page 7

Task 1: Preconversion Cleanup

Before starting a conversion it is important to analyze, clean up, and possibly rewrite and retag the content so it "fits" better into the DITA model. It is best to do as much cleanup before rather than after conversion.

It's easy to say "rewrite and retag to fit the DITA model," while actually doing that can be quite time consuming. However, it really is best to take as much time as you can to do this work before converting to DITA. Doing this work post-conversion will be even more difficult and time consuming.

You'll need to do the following:

- Clean up inconsistently or improperly tagged content. Any formatting overrides assigned to text or paragraphs will be lost in the conversion.

  If this formatting is important to preserve, you must make sure that each unique "style" is defined as a character or paragraph tag. This is frequently

a problem with text ranges that have been bolded or italicized with the "B" or "I" buttons.

The FM2DITA Tag Cleanup command is designed to help with character tagging inconsistencies.

The FM2DITA Untag Boundary Spaces command ensures that all spaces between tagged text ranges have no character tag applied. XML whitespace normalization rules will typically cause leading and trailing whitespace in an element to be removed, so if an inline element starts or ends with whitespace, that will often be lost in a conversion.

- Clean up inconsistent use of conditional tagging.

  It is difficult to transfer conditional tagging to DITA's filtering attributes. First, for this to map properly, the start/end points of a condition must align with the start/end tag of the corresponding element. Because this unlikely to work well for inline tagging (even if you are able to align the start/endpoints), you'll end up with less than ideal DITA content, so it is best to assume that all conditional tagging should be applied at the paragraph level.

  The FM2DITA Condition to Attribute command transfers conditions that exist at the start of a paragraph to the corresponding element(s). By default the condition names are assigned to the element's product attribute, but you can specify a different default and can define specific mapping of condition to attribute names.

  The FM2DITA Rename Conditions command can be used before conversion to globally rename conditions if the current condition names are not appropriate to be used as attribute values going forward.

  The FM2DITA Show All Conditions command will show all conditions in the book or file. It is always best to show all conditions in all files before performing a conversion.

- Rewrite your content so it's consistent and fits into the parts of the DITA model you plan to use.

  If you want your topics to include a short description (the DITA <shortdesc> element), make sure every heading that defines a new topic is followed by a paragraph to be used as the short description.

  To increase the possibility for topic reuse, it's suggested that you reduce the number of inline cross-references. If possible, consider moving all of your cross-references into relationship tables; conversion time is an ideal time to make that move.

- Retag the paragraph styles so that the content in the various topics is clearly and unambiguously identified. It's better to have too many paragraph tags than too few.

  If you are converting from chapter-based FM files into multiple topic types, you'll likely need to develop a tagging scheme with new style names that allow you to mark sections of content as belonging to specific sections within each DITA topic model. This is particularly important for the task model, which contains multiple sections with similar types of content; the only way for the conversion table to distinguish between a "prereq" or "context" paragraph is through the paragraph tag name.

  The FM2DITA Retag Paras command can be used to assign prefixes to existing paragraph tag names within certain sections. This command can make multiple passes on the content to build up tag names which create unique, section-based tag groups, that allow a conversion table to create properly structured DITA topics.

  Similarly, the FM2DITA Retag Tables in Paras command will assign prefixes to table format names within certain paragraph tags. This can help to ensure that tables get wrapped in the proper elements.

A brief note about Index marker conversion:

- If your content contains Index markers (converted to fm-indexterm elements while in FM), the export API client associated with the structured application imported into the files in Task 3, is responsible for converting from FrameMaker marker syntax to the corresponding DITA markup. If those Index markers define "See" and/or "See also" entries, the conversion to the proper DITA elements is dependent on the text in those markers matching up with the syntax expected by the API client. If you're using DITA-FMx, make sure that the character tags used to format the entry match those in the Index Options dialog (in DITA Options). If you're using another export API client, you should do some testing to make sure the Index markers will convert as expected. (Or just assume that you'll fix them in a post-conversion pass.)

Once your files have been properly cleaned up and tagged, you're ready to start the conversion process.

TASK

1. Create a folder named *conversion* where the conversion process will be performed. Create *source* and *out* folders within the *conversion* folder.
   - These folder names can be anything that makes sense for your work-flow.
2. Copy the source FrameMaker and any auxiliary files (graphics, etc) to the *source* folder.
   - Only references to files that are converted at the same time will be resolved. If you have references to other books, those will need to be fixed in a post-conversion pass.
   - If you are converting multiple FM files and there is no book file, create one. This isn't required, but it's easier to run the automation commands on a book rather than each file individually.
3. Create and set up an *fm2dita.ini* file for this project. This INI file should be created in the *conversion* folder.
   - When an FM2DITA command is run on any FM or BOOK file, the current folder is checked for an *fm2dita.ini* file. If that file is not found, the parent folders are scanned (allowing you to maintain just one INI file per project). If that file is not found, the default *fm2dita.ini* file (in the Pubs-Tools folder) is used.
4. Open the book file.

5. Delete any frontmatter, backmatter, and other generated or non-content files.
   - The only files that should remain in the book are "content" files. The conversion table is typically not set up to support cover pages or other frontmatter and backmatter.
6. Open all files in the book.
   - This isn't required, but can simplify problems that are caused by missing graphics or unresolved references. If you don't open the files, the FM2DITA commands will open each file, then save it before closing.
   - Place focus on the book, press **Shift** and click **File > Open All Files in Book**
7. Convert any text insets to text.
   - If any exist, locate each inset, double-click, select **Convert to text**, repeat. The FM2DITA conversion process is currently not set up to support text insets (although this may be supported in the future).
8. Perform any necessary preconversion processing.
   - If you are using any of the FM2DITA preconversion tools, make sure that the *fm2dita.ini* file has been set up properly and copied to the *source* folder.
9. Resolve all cross-references.
   - Any unresolved references will not be resolved in the XML files.
   - With focus on the book, use the **Edit > Update References** command.
   - Also make sure all graphics are "found". You can convert with missing graphics, but it'll likely interrupt the conversion process at various points.
10. Confirm that all user variables, cross-reference formats, table formats, and styles are included in the conversion table.
    - Place focus on book, choose **Pubs-Tools > FM2DITA > Catalog Report**
    - Compare the results to previous catalog reports. If new objects or styles exist, add them to the conversion table.
11. Save and optionally close all files in the book.
    - Place focus on book, press **Shift** and click **File > Save All Files in Book**
    - Place focus on book, press **Shift** and click **File > Close All Files in Book**
12. Optionally save the *source* folder to a new name (*source-1*?) to make it easier to roll back to this point in case something goes wrong.

RELATED INFORMATION:

"Typical Conversion Process" on page 6
"Task 2: Structured Conversion" on page 12

Task 2: Structured Conversion

Creates structured BOOK and FM files from the unstructured source files.

After completing any preconversion cleanup, the next step is to apply the conversion table to the documents.

NOTE: *For FrameMaker 7.2 users, menu items indicated on the* ***StructureTools*** *menu can be found on the* ***File*** *menu.*

TASK

1. Open the book and all files from Task 1: Preconversion Cleanup.
2. Apply the conversion table to the book.
   - Open conversion table file
   - Place focus on the book
   - Choose **StructureTools > Utilities > Structure Current Book**. In the dialog, select the conversion table from the list and specify the output folder as *out* subfolder (created in Task 1). Select to overwrite the old files.
   - FrameMaker Log may report errors, check the console, probably nothing useful or important though.
   - This creates structured FM binary files from the unstructured files.
3. Close the original unstructured book and files. Don't save when closing.
4. Open all structured files.
   - Place focus on the "structured" book, press **Shift** and click **File > Open All Files in Book**
5. Review all files, verify that all have a <dita> root element.
   - If some files don't have a <dita> root, determine what didn't convert properly. You'll either need to fix some invalid tagging in the source file or possibly update the conversion table. Fix things and start over at step 2.
   - Note that you can't "validate" the files at this point since there is no structured application assigned to the files.
   - If the conversion table was recently modified or you are processing a new book, review extra the first time. Check the resulting structure to make sure everything is getting wrapped and tagged properly.
6. Save and optionally close all files in the book.
   - Place focus on book, press **Shift** and click **File > Save All Files in Book**
   - Place focus on book, press **Shift** and click **File > Close All Files in Book**
7. Optionally save the *out* folder to a new name (*out-1*?) to make it easier to roll back to this point in case something goes wrong.

RELATED INFORMATION:

"Typical Conversion Process" on page 6
"Task 1: Preconversion Cleanup" on page 7
"Task 3: Conversion Processing" on page 13

Task 3: Conversion Processing

The bulk of the FM2DITA automation happens here.

PREREQUISITES:

Verify that the *fm2dita.ini* file is set up properly. Refer to the FM2DITA documentation for details on the commands you plan to use.

- Now is the time to carefully consider the topic and map file naming scheme. You'll need to set the file name template value (TopicNameTpl parameter) before starting this task. Once this parameter has been set it should not be changed without starting over at this point. Similarly, the MapNameTpl parameter controls the file naming for generated maps.
- If you're not using the FM2DITA structure application, make sure that the structure application you are using supports all of the topic models you plan to generate. This typically means that your app should be based on the ditabase model.

NOTE: *Between each step or as needed, remember to save all files.*

TASK

1. Open the book and all files from Task 2: Structured Conversion.
2. Place focus on the book file, choose **Pubs-Tools > FM2DITA > Import Template and EDD**.
   - This imports the template and EDD associated with the structure application specified by the TopicAppName parameter in the *fm2dita.ini* file.
3. If the file naming relies on chapter numbering, set up chapter numbering in the book.

4. With focus on the book, choose **Pubs-Tools > FM2DITA > Check for Topic Collisions**
   - If collisions are reported, fix the title text before continuing.
   - If filenames are based on topic IDs, you'll need to run **Assign IDs to Topics** before running this command.
5. Run the following FM2DITA commands on the book in the order shown.
   - Be sure to re-focus the book with each command and watch the console for errors.
   - Review the documentation to decide which commands are useful for your processing. In general it should be fine to run a command that isn't needed.
   - The order of the commands is important. Run commands higher on the menu before those lower on the menu.
   - **Assign IDs to Topics in Book** - *Required for all conversions*
   - **Unwrap Elements in Book**
   - **Delete Elements in Book**
   - **Condition to Attribute in Book**
   - **Fix Images in Book**
   - **Fix Tables in Book**
   - **Fix Cross-refs in Book**
   - **Map Hypertext Markers in Book**
   - **Related Links to Reltable in Book**
   - **Flatten Cross-refs in Book**
   - **Move Markers in Book**
   - **Variables to Conrefs in Book**
   - **Merge Code Lines in Book**
   - **Tab to Spaces in Book**
   - **Delete Invalid Attributes in Book** - *Typically required to delete the Id and Idref attributes*
   - **Delete Unstructured Markers in Book** - *Typically required to delete Cross-Ref markers*
   - **Delete Empty Elements**
6. Save and optionally close all files in the book.
   - Place focus on book, press **Shift** and click **File > Save All Files in Book**
   - Place focus on book, press **Shift** and click **File > Close All Files in Book**
7. Optionally save the *out* folder to a new name (*out-2*?) to make it easier to roll back to this point in case something goes wrong.

RELATED INFORMATION:

"Typical Conversion Process" on page 6
"Task 2: Structured Conversion" on page 12
"Task 4: Pre-export Validation" on page 15

Task 4: Pre-export Validation

All files should be valid before exporting to XML.

Before saving FM binary files to XML, they must be valid. Saving an invalid file to XML may work (with errors reported), but it may also cause FrameMaker to crash. It is always best to do as much cleanup as possible in the FM binary files before saving to XML.

TASK

1. Open the book and all files from Task 3: Conversion Processing.
2. Validate each file in the book.
   - For each document in the book, use **Element > Validate** to find and fix any problems.
   - If you run into problems that can't easily be fixed and want to reconvert a single file, that should be fine; you don't need to reconvert the entire book.
   - It's OK that the book contains the element of NoName at the root.
3. Carefully review files for errors like empty paragraphs or other issues that are valid but may result in less than ideal structure.
   - It is much easier to review as chapter files than after breaking into topics.
4. Save and optionally close all files in the book.
   - Place focus on book, press **Shift** and click **File > Save All Files in Book**
   - Place focus on book, press **Shift** and click **File > Close All Files in Book**
5. Optionally save the *out* folder to a new name (*out-3*?) to make it easier to roll back to this point in case something goes wrong.

RELATED INFORMATION:

"Typical Conversion Process" on page 6
"Task 3: Conversion Processing" on page 13
"Task 5: Export Maps and Topics" on page 16

Task 5: Export Maps and Topics

Generates an XML file for each "topic" in each chapter and creates the maps for the project.

The export process used by the **Write XML Topics** command, relies on the structured application imported into the files (as a template and EDD) in Task 3. In particular, the read/write rules and export API client associated with the structured application will control how elements are mapped and converted to XML. If you're using a structure application other than the FM2DITA app, you should do some testing to make sure the elements will export properly to XML. This can be done by just using a **Save As XML** command, then checking the resulting XML file for errors.

TASK

1. Open the book and all files from Task 4: Pre-export Validation.
2. With focus on book, choose **Pubs-Tools > FM2DITA > Write Root and Chapter Maps**.
   - If you just want a single map rather than root and sub maps, use the **Write Single Map** command instead.
3. With focus on book, choose **Pubs-Tools > FM2DITA > Write XML Topics**.
   - This process may take a long time; you'll be given an estimate of time based on the number of topics (this may not be terribly accurate).
   - Before writing XML topics form the file, it is saved, then after processing it is closed without saving. This allows you to go back and re-write XML files later if needed.
   - If any files remain open, it's likely that there was a problem and the process stopped mid-book. Review the open files and check for errors in the console.
4. Close the book.

AFTER COMPLETING THIS TASK:

If the **Related Links to Reltable** command was used, this created separate maps with relationship tables. You'll either need to manually add references to these reltable maps or copy/paste the reltables into the root or chapter maps.

RELATED INFORMATION:

"Typical Conversion Process" on page 6
"Task 4: Pre-export Validation" on page 15
"Task 6: Test Exported Files" on page 17

Task 6: Test Exported Files

After generating the DITA XML files, it is important to perform some level of validation to ensure that everything converted properly.

It can be very time consuming to review every XML file that is generated, especially for a very large project. At a minimum, review each "type" of file. It's likely that common structures will convert in a similar manner.

TASK

1. Open the root map.
   - All topicrefs should resolve (no "MISSING FILE" messages).
2. Open each submap.
   - All topicrefs should resolve (no "MISSING FILE" messages).
3. Open random topics.
   - Look for problems, make sure all's well.
4. Run the map through a DITA-OT build as a sanity check.
5. If all looks good, set up the root map as bookmap.
   - Retag the root element to bookmap.
   - Retag topicrefs to chapter and appendix elements.
   - Set up frontmatter/backmatter elements.
6. Optionally, run map through a DITA-FMx book-build.

RELATED INFORMATION:

"Typical Conversion Process" on page 6
"Task 5: Export Maps and Topics" on page 16

Conversion Table Development

Tips and guidance on creating conversion tables.

The fundamental concepts of setting up a conversion table are fairly basic, but actually doing so can be quite time consuming and complex. The best place to find all of the details on this process is the *Structure Application Developer's Guide* from Adobe. Not much has changed over the years on this subject, so you should be able to use most any version of this document with any version of FrameMaker. With FM7.2, this PDF was installed in the *OnlineManuals* folder,

for later versions you'll have to locate it online. Just search for the document title.

The FM2DITA preconversion commands are used before applying the conversion table, but two commands, Retag Paras and Retag Tables in Paras, will affect the way the conversion table is developed. The FM2DITA sample files include a sample conversion table, we recommend that you take a look at this conversion table, so you'll be familiar with a simple, but functional conversion table.

RELATED INFORMATION:

"Initial Conversion Table Setup" on page 18
"Setting Up Mapping Rules" on page 19
"Working with Qualifiers" on page 22
"Setting Up Wrapping Rules" on page 23
"Conversion Table Tips" on page 26

Initial Conversion Table Setup

FrameMaker can create the basic conversion table for you to use as a starting point.

The first step in creating a conversion table is to run the **StructureTools > Generate Conversion Table** command. This creates an initial conversion table based on the styles and objects in the current file. In order to ensure that the conversion table supports all of the styles and objects in all files in a book, you need to create a file that contains examples of all styles and objects used in the book. The easy way to do this is to copy and paste the content from each file into a single file. Once you've got this merged file, run the Generate Conversion Table command, and you'll have a good starting point for your conversion table.

The conversion table generated by the Generate Conversion Table command isn't terribly useful as-is. You can apply it to a file or the entire book, but it'll just create a nonsense structure that uses tag names that match your style and object names. The first column in the conversion table specifies the styles and objects being mapped, the second column is the element that each style or object is mapped to, and a third column (empty by default) is for an optional qualifier (discussed later). You add additional rows to the table that define element wrapping rules that let you build up the desired structure.

The table that's created by this command will start with rows for paragraph style mapping (identified by a "P:" prefix) then rows for character styles ("C:"). After that will be rows that map other objects like cross-references ("X:"), markers ("M:"), user variables ("UV:"), and graphics ("G:"). All of these prefixes will be followed by the associated style or object name, except for the "G:" which just maps to all graphics. You'll also see rows for tables and table "parts." Each table format will be identified with a "T:" prefix, and then you'll see codes with no

descriptive text for each table part: “TT:” (table title), “TH:” (table head), “TB:” (table body), “TF:” (table footer, which isn’t valid in DITA), “TR:” (table row), and “TC:” (table cell).

Before doing anything serious, you may want to do is to modify the formatting and layout of the table. We suggest reducing the font size and expanding the table width by reducing the margins, then possibly adding another column for notes. This is all optional, but you’ll be spending a lot of time in this table, so make it something that’s easy remember.

A conversion table may be broken into multiple tables to make it easier for you to organize the processing rules, as defined by each row in the table. In general, there are two fundamental groups of rules, mapping rules and wrapping rules. The mapping rules create the base element from a style or object, and wrapping rules will wrap existing elements in more elements.

RELATED INFORMATION:

"Conversion Table Development" on page 17
"Setting Up Mapping Rules" on page 19
"Working with Qualifiers" on page 22
"Setting Up Wrapping Rules" on page 23
"Conversion Table Tips" on page 26

Setting Up Mapping Rules

Mapping rules define the initial mapping from styles and objects to elements.

The first task is to organize the rows in the table so they are ordered by some reasonable logic. We typically put the heading styles at the top, and order them from highest to lowest levels. The order really doesn’t matter, it just helps to be able to locate things later on. If you’re converting to multiple “models” (task, concept, reference), it’s best to group those styles together in the table. You’ll probably have common styles that are used for multiple models and other styles that are specific to a style. Group the common styles first, then the model-specific styles. Feel free to add empty rows between groups to help organize things. You can also split the table into multiple smaller tables and add heading and descriptive text before each table to help document the process. For smaller tables this isn’t very important, but as the tables grow in size and complexity, this can be useful.

Once you’ve made a first pass at organizing the styles and objects in the table, start assigning initial elements to those styles and objects. Keep in mind that you’re defining a process of element wrapping to build up your final structure from the lowest level to the top. In order to do this you must be very familiar with the structure of DITA (or whatever is your structural model). If you don’t have a very good understanding of the model, stop and learn that before continuing.

Typically, the paragraph styles will map to a <p> element. Since in DITA, you'll end up wrapping <p> in an <li> for list items or <note> for notes, this is a safe place to start. Headings will map to a <title> element. If you do have a specific paragraph style that maps to the DITA <shortdesc> element, go ahead and map that to <shortdesc> instead of <p>. If you've got definition list styles, and one of those is the "term", you'll probably want to map that to the <dt> element. If you want to know "from whence" the new elements came, or if you need to be able to apply special formatting to certain elements, you can assign a value to the outputclass attribute. In the following partial example, there are two rows, the first just maps the "Body" paragraph style to the <p> element, and the second maps the "BodyIndent" to a <p> element with an outputclass attribute value of "BodyIndent".

P:Body	p	
P:BodyIndent	p[outputclass="BodyIndent"]	

You'll also typically want to start assigning values to the qualifier column (the third column), but we'll discuss that a bit later when we get into element wrapping.

Character styles will often not be wrapped more than once. The initial mapping from style to element is typically all that's needed. One exception to this is if you're able to wrap sequential <uicontrol> elements in a <menucascade> element. If you just use "Bold" for all types of bold tagging regardless of their semantic nature, you'll probably just have to map that to the <b> element, but if you do have semantically named character styles, it's nice to map them to corresponding DITA elements.

After the paragraph and character style mapping, are various object mapping rows. Variables use the "UV:" (user variable) and "SV:" (system variable) prefix, followed by the variable name. If a variable isn't included in the conversion table, it will convert as plain text. You'll typically not map system variables to any elements. If you want your user variables to convert into conrefs (via the FM2DITA Variable to Conref command), you'll need to map the variable to an inline element (typically <ph>, but could be anything that makes sense) and include a conrefid attribute. This conrefid attribute is not a valid DITA attribute, it's just what is used by the Variable to Conref command to identify elements to turn into conrefs.

All cross-ref formats must be included in the conversion table. If a cross-ref is not included, you'll end up with an element named <CROSSREF> in the resulting file. Cross-refs are identified by a "X:" prefix and include the format name. These should be mapped to the <xref> element, and you'll typically set the outputclass attribute to the value of the cross-ref format name. This allows the formats to round-trip and be assigned the proper format name. If you do not

want a cross-ref to convert as a "formatted" reference, omit the outputclass attribute.

All markers that you want to convert to DITA elements, must be listed in the conversion table. Any marker not listed, will initially convert as an unstructured marker, which is represented as a processing instruction (PI) in XML. You'll want to delete these unstructured markers before saving to XML, otherwise they will persist as PIs and cause unnecessary bloat in the XML. In general, you'll just want to preserve Index markers. If you're using DITA-FMx, it's possible to convert other markers into an <fm-data-marker> element (which saves as a DITA <data> element).

The "G:" code in the first column of a conversion table will map all graphics to the specified element. There's no way to identify different types of graphics, the are all just mapped through this one row. For DITA, you'll map graphics to the <image> element. Graphic and table mapping can make use of a special "promote" operator. When used for graphics, this breaks the image out of the current paragraph and makes it a child of the "parent" element. While this may seem like a good thing to do, use of this feature will remove inline images from the container paragraph (which makes them no longer inline). The best practice for images is to always anchor the anchored frame to an "image-anchor" paragraph or to a "figure-title" paragraph. If this is done, you should not use the "promote" operator for graphics.

Tables are identified with the "T:" prefix followed by the table format name. You can just use the "T:" code without the table names to map all tables to the same element. If you don't want to preserve the table formats, that is fine. Otherwise, if you do want tables to continue to use different formats, provide a row for each table format, and map each to the <tgroup> element. An important point to understand is that the table "object" in a FM document, maps to the <tgroup> element in a DITA table. The <table> element is a "container" that wraps the table object. Where this can cause some confusion is when working with a table title. The DITA <title> element is a child of the <table> element. This means that it is not in the location where a FrameMaker table object expects its title to be. FrameMaker expects the table title to be inside of the table object, which means it would be a child of the <tgroup> element. Because of this you never want to use the FrameMaker Table Designer to enable the table title, this will create an invalid structure in DITA.

Following the table format mappings you'll map each of the sub-table objects (table parts) to the appropriate elements. The "TT:" code maps the table title, which would typically be a <title> element. The "TH:" and "TB:" codes map the table head and table body, which would be <thead> and <tbody> elements. FrameMaker tables can have a table footer area which is the "TF:" code. DITA does not support the table footer, so if you map this to the <tfoot> element, and you use the FM2DITA Fix Tables command, it will merge all "tfoot" rows to the end of the "tbody" rows and assign a special outputclass value so you can format

them differently. Table rows and table cells use the "TR:" and "TC:" codes, which should map to the <row> and <entry> elements.

If you want to convert tables into other table types (like <simpletable> or <choicetable>), map the appropriate table format to those table element types. (These are the actual table objects, equivalent to a <tgroup>.) You'll end up with a <simpletable> that has child elements like <thead> and <tbody>, which isn't valid DITA, but since there's only one option for mapping the sub-table objects, this is what's done. The FM2DITA Fix Tables command will locate these alternate table types and retag the internal structure to make them into proper DITA structures. If you have custom table structures, you can define them in the AltTableTypes section of the *fm2dita.ini* file.

As mentioned with graphics, tables can make use of the "promote" operator to break the table out of the current element and elevate it to a sibling element. This it typically the right thing to do since tables are typically anchored into the previous paragraph.

RELATED INFORMATION:
"Conversion Table Development" on page 17
"Initial Conversion Table Setup" on page 18
"Working with Qualifiers" on page 22
"Setting Up Wrapping Rules" on page 23
"Conversion Table Tips" on page 26

Working with Qualifiers

For all but the simplest of conversions, you'll need to use qualifiers to properly identify and wrap like named elements used in different structures.

Qualifiers are labels that are assigned to a mapping or wrapping rule to allow you to distinguish the resulting element from other like-named elements in later wrapping rules. Qualifiers can also be used to assign labels to multiple different types of elements so you can refer to groups of elements without naming each type in later wrapping rules. If used, a qualifier label is included in the third column of a mapping or wrapping rule.

There are no fixed rules about how to naming conventions for qualifiers. As you develop conversion tables, you'll likely come up with your own naming conventions that make sense to you and work best for your content. We'll describe one method here, but the best method is the one that "works."

The best way to understand how qualifiers work is to review a working conversion table. The FM2DITA sample files include a conversion table that has been fully annotated (in the fourth column). We recommend that you read through that file.

...

RELATED INFORMATION:
"Conversion Table Development" on page 17
"Initial Conversion Table Setup" on page 18
"Setting Up Mapping Rules" on page 19
"Setting Up Wrapping Rules" on page 23
"Conversion Table Tips" on page 26

Setting Up Wrapping Rules

Wrapping rules are used to wrap one or more elements in new elements to create hierarchies in the structured file.

For all but the simplest of conversions, and definitely for DITA, after creating the initial base elements with the mapping rules, you'll need to define rules that wrap those elements in other elements to create nested structures. These rules can be created in the same table as the mapping rules, or you can create additional tables in the same file to contain the wrapping rules.

The format for a wrapping rule is similar to that of a mapping rule in that the first column defines the element(s) that are acted on, or wrapped, the second column defines the element that is the container or wrapper, and the third column defines the optional qualifier. The first column may be very lengthy in its description of exactly what sequence of elements to match on and wrap. The second column is always just the wrapper element name and possibly attribute(s) that are assigned to that element.

The syntax of the first column is similar to that of an EDD (or DTD) general rule syntax. It describes a sequence of one or more elements, which if met, will cause those elements to be wrapped. To identify an element you use the "E:" prefix, which is followed by the element name and/or a qualifier label. You can specify a sequence of elements by separating them with a comma, and if you want to specify optional groups of elements use parenthesis to define the group and the vertical bar character to separate the elements within the group. Special frequency operators follow an element or group to indicate the number of times that item can exist. The options are "*" (zero or more), "+" (one or more), and "?" (zero or one). If no frequency operator is provided, that means the element must exist once.

The following simple example performs these wrapping rules:

- select a sequence of one or more <li> elements and wrap them in a <ul> element
- select a sequence of <p> and/or <ul> elements and wrap them in a <body> element

- select a sequence of a <title> element optionally followed by a <shortdesc> element followed by a <body> element and wrap them in a <topic> element
- select a sequence of one or more <topic> elements and wrap them in a <dita> element

E:li+	ul	
(E:p | E:ul)+	body	
E:title, E:shortdesc?, E:body	topic	
E:topic+	dita	

While these rules will "work", it's not a very realistic wrapping table for a number of reasons. First, it assumes that all topics are defined by the same heading level; there's no topic hierarchy created by these rules. Also, it assumes that the content in the topics only consist of paragraphs and bulleted lists (and these bulleted lists only have one paragraph per list item).

A possibly more realistic example would be one that makes used of qualifiers. The following examples show a mapping table that uses qualifiers and a wrapping table that makes use of those qualifiers.

First, a partial mapping table example. These mapping rules assign qualifiers for the different heading levels so this hierarchy can be preserved. No qualifier is assigned to the <shortdesc> element since this is a unique element, only used in one situation. The "Body" tag is mapped to a <p> element and assigned a "L0" (level 0) qualifier. You could have additional types of paragraphs at this level and give them all "L0" qualifiers, and they would all get grouped together in the wrapping rule. The "Bullet1" tag is mapped to a <p> element and assigned a "B1" qualifier. This allows you to have list items that have multiple paragraphs, with the "Bullet1Cont" tag using a "L1" (level 1) qualifier.

P:Title	title	H0
P:Heading1	title	H1
P:Heading2	title	H2
P:Info	shortdesc	
P:Body	p	L0
P:Bullet1	p	B1
P:Bullet1Cont	p	L1

Next, a partial wrapping table example makes use of the qualifiers defined above. The list items always start with a “B1” item (note that the element name is not needed when referring to a “class” of elements), and are followed by zero or more “L1” items (in this case it’s only the Bullet1Cont paras, but could be other items like nested lists or figures, etc.). The “B1” elements (one or more) are wrapped in a <ul>, which is assigned a “L0” qualifier. Then, all “L0” items are wrapped in <body>. Note that using qualifiers can simplify the wrapping rules by allowing you to refer to a group of elements. This also makes it easier to add new elements in the future as long as you remember the “rules” you used for the qualifiers.

In this example, when wrapping topics, because we assigned qualifiers to each of the heading levels, we are able to maintain the original hierarchy. Start at the lowest level (“H2” in this case) and wrap all <title>, <shortdesc> and <body> sequences in a <topic> element (which is assigned a “T2” qualifier). Then go to the next level (“H1”) and wrap those <title>, <shortdesc>, <body>, and zero or more “T2” <topic> sequences in a <topic> (assigned a “T1” qualifier). Then one more level to get the “T0” topics. It’s not likely that a file with have multiple “T0” topics, but just in case, make the final rule inclusive enough to wrap up more than one in a <dita> element.

E:[B1], E:[L1]*	li	B1
E:li[B1]+	ul	L0
E:[L0]+	body	
E:title[H2], E:shortdesc?, E:body?	topic	T2
E:title[H1], E:shortdesc?, E:body?, E:topic[T2]*	topic	T1
E:title[H0], E:shortdesc?, E:body?, E:topic[T1]*	topic	T0
E:topic[T0]+	dita	

The root <dita> element may not always be needed, but for the FM2DITA topic processing tools, this is required.

...

RELATED INFORMATION:

"Conversion Table Development" on page 17
"Initial Conversion Table Setup" on page 18
"Setting Up Mapping Rules" on page 19
"Working with Qualifiers" on page 22
"Conversion Table Tips" on page 26

Conversion Table Tips

TIP: *This topic will be updated over time.*

<note> elements

The common note types require just one attribute value (or no attribute value in the case of a plain "note"). For these note types, just set the type attribute to the appropriate value. The following example creates a "tip" note:

```
note[type="tip"]
```

However, if you need to create a note that uses a type that's not supported, you have to set the type attribute to "other" and the othertype attribute to the desired note type. The following example creates a "warning" note:

```
note[type="other" & othertype="warning"]
```

RELATED INFORMATION:

"Conversion Table Development" on page 17
"Initial Conversion Table Setup" on page 18
"Setting Up Mapping Rules" on page 19
"Working with Qualifiers" on page 22
"Setting Up Wrapping Rules" on page 23

Editing the fm2dita.ini File

The fm2dita.ini file defines the configuration settings for all of the FM2DITA commands.

The *fm2dita.ini* file controls many of the default settings for the FM2DITA processing. A default version of this file should be created in one of the following locations:

- Windows XP: <*program files*>*Adobe**FrameMaker*<*ver*>*PubsTools**fm2dita.ini*
- Windows Vista, 7, 8, or 10: <*appdata*>*Adobe**FrameMaker*<*ver*>*PubsTools**fm2dita.ini*

To locate this folder, use the **Pubs-Tools > Open Pubs-Tools Folder** command.

If this file exists in the same folder as the FM file or book being processed, that INI file will be used instead of the default file. This allows you to maintain separate INI files for each project.

[General]

MainFlow

Overrides the default flow-detection and processes the specified flow. In most cases this should not be specified, but if you want to process a flow that's not recognized as the main flow, try setting it here.

Default: (nothing)

Used by: most commands

AssignIdElems

Space-delimited list of elements (element names) to assign IDs. If empty, IDs are assigned based on element definitions in the EDD where the @id attribute is identified as required.

Default: (nothing)

Used by: Assign IDs to Topics, Fix Cross-refs

TopicElems

Space-delimited list of "topic" elements. Leave empty to check the EDD for element definitions where the @class attribute contains "topic/topic".

Default: (nothing)

Used by: Assign IDs to Topics, Fix Cross-refs

AttributeDisplay

Specifies the attribute display setting applied by the Import Template and EDD command.

Valid values: None, ReqSpec, All

Default: `ReqSpec`

Used by: Import Template and EDD

IdPrefix

String prefix for auto-generated IDs.

Default: `id`

Used by: Assign IDs to Topics

IdType

Specifies the type of auto-generated IDs. Available types are GUID (globally unique ID) or QUID (quasi-unique ID, shorter FM-based values).

Valid values: GUID, QUID

Default: `GUID`

Used by: Assign IDs to Topics

TopicNameTpl

Template string for generated topic file names. Should include building blocks as well as the required file name extension (typically ".xml" or ".dita"). Do not specify a path. For building block details, see Topic and Map Template Building Blocks.

Default:
`<$FM_CHAPNUM[L]>-<$TITLE_NOSPACECAMELLOW>.xml`

Used by: Fix Cross-refs, Check for Topic Collisions, Write Chapter Map, Write XML Topics, Write Root Map, Write Root and Chapter Maps, Write Single Map

MapNameTpl

Template string for generated chapter map file names. Should include building blocks as well as the required file name extension (typically ".ditamap"). Do not specify a path. For building block details, see Topic and Map Template Building Blocks.

Default:
`<$FM_CHAPNUM[L]>-<$TITLE_NOSPACECAMELLOW>.ditamap`

Used by: Write Chapter Map, Write Root and Chapter Maps

RootMapNameTpl

Template string for generated root ditamap file name. Should include building blocks as well as the required file name extension (typically ".ditamap"). Do not specify a path. For building block details, see Topic and Map Template Building Blocks.

Default: `_!<$FM_FILENAME[L]>.ditamap`

Used by: Write Root and Chapter Maps

RootMapIsBookmap

Controls the type of root map created. If set to "1", a bookmap will be written for the root map, otherwise a "regular" map is written.

Default: `0`

Used by: Write Root Map, Write Root and Chapter Maps

DeleteRelLinks

Specifies if related-links are deleted while writing XML topics. Valid values are 0 (don't delete) or 1 (delete)

Default: `1`

Used by: Write XML Topics

FmXrefAttr

The xref element's attribute to which the FM cross-ref format is assigned (for fm-xref usage). Typical values are "type" or "outputclass". If set to nothing, the format name is not assigned to the xref element.

Default: (nothing)

Used by: Fix Cross-refs

FlattenXrefs

if set to "1", tells the Fix Cross-refs command to perform the "flatten cross-ref" command as well. This is used to revert the functionality back to the original single command.

Default: `0`

Used by: Fix Cross-refs

TopicAppName

Topic application name as registered in the structure application definition file.

Default: `FM2DITA-Topic-1.1`

Used by: Variables to Conrefs, Write XML Topics

DefaultCondToAttr

Filtering attribute name to use for conditions not mapped in the CondToAttrMap section of the INI file.

Default: `product`

Used by: Condition to Attribute

CheckParentCond

Specifies whether to apply a filtering attribute if set on element's parent. Valid values are 0 (don't check parent) or 1 (check parent).

Default: `0`

Used by: Condition to Attribute

StopWords

Space-delimited list of words to exclude from topic titles when generating file names as well as strings for the `$TITLE*` building blocks. This can also specify a file name (plain text) which contains the list of stop words (one word per line). The file name can be an absolute path, or relative (if it starts with "./" or "../"). If relative, it is relative to the INI file. If this

parameter is commented out or set to no value, no stop words will be stripped.

Default: (nothing)

Used by: Fix Cross-refs, Check for Topic Collisions, Write Chapter Map, Write XML Topics, Write Root Map, Write Root and Chapter Maps, Write Single Map

PunctuationChar

Specifies the character used to replace punctuation when strings are returned by the `$TITLE*` building blocks. If not present or set to an empty string (nothing), this punctuation is stripped from the return string.

Default: (nothing)

Used by: Fix Cross-refs, Check for Topic Collisions, Write Chapter Map, Write XML Topics, Write Root Map, Write Root and Chapter Maps, Write Single Map

ConrefLibrary

File name for conref library. Do not specify a path or use building blocks.

Default: `shared.xml`

Used by: Variables to Conrefs

TabToSpaces

Number of spaces to replace for each tab character.

Default: `4`

Used by: Tab to Spaces

DeleteAttrs

Space-delimited list of attributes to delete. If empty (default), you'll be prompted for values.

Default: (nothing)

Used by: Delete Invalid Attributes

DeleteMarkers

Space-delimited list of unstructured markers to delete. If empty (default), you'll be prompted for values.

Default: (nothing)

Used by: Delete Unstructured Markers

UnwrapElems

Space-delimited list of elements to unwrap. If empty (default), you'll be prompted for values. This value can be the element name or an element/attribute specification (like b/@outputclass='unwrapme').

Default: (nothing)

Used by: Unwrap Elements

DeleteElems

Space-delimited list of elements to delete. If empty (default), you'll be prompted for values. This value can be the element name or an element/attribute specification (like b/@outputclass='deleteme').

Default: (nothing)

Used by: Delete Elements

MergeElems

Space-delimited list of elements to merge. If empty (default), you'll be prompted for values.

Default: (nothing)

Used by: Merge Code Lines

DeleteEmpty

Space-delimited list of elements to delete. If empty (default), you'll be prompted for values.

Default: (nothing)

Used by: Delete Empty Elements

MoveTblOutputclass

If set to "1", specifies that the outputclass value is moved from the tgroup element to the parent table element.

Default: `0` (outputclass is not moved)

Used by: Fix Tables

SaveTableWidth

If set to "1", saves the relative table width (as a percentage) to the parent table element's pgwide attribute. The percentage is rounded to the nearest higher 5.

Default: `0` (don't save table width)

Used by: Fix Tables

SaveTableShading

If set to "1", adds a value to the cell element's outputclass attribute based on the shading applied to the cell. This value includes the color name and the fill pattern number (a value from 0-15). For example, if the color is blue and the fill is a dotted pattern, the value may be "fill-blue-4".

Default: `0` (don't save table shading)

Used by: Fix Tables

TableFmtPrefixChopChar

Specifies the character to use to determine the prefix to remove from table formats applied to each table. For example, if your table format is "con-FormatA", but you want it to be just "FormatA", specify the "-" character in this parameter.

Default: (nothing)

Used by: Fix Tables

UnwrapTables

A space-delimited list of table formats to unwrap (convert to text). If empty (default), you'll be prompted for values.

Default: (nothing)

Used by: Tables to Text

UnwrapTablesPrefix

A space-delimited list of prefixes to apply to the paragraphs that result from the unwrapping of tables.

Alternatively, a complex mapping option is available that allows column-based and/or tag based assignment of the new tag names. See Complex mapping option for details.

Default: (nothing)

Used by: Tables to Text

IncludeMetadataInMaps

If set to "1", includes topicmeta/critdates/created/@date='*YYYY-MM-DD*' in generated maps, otherwise no metadata is included.

Default: `1` (include metadata)

Used by: Related Links to Reltable, Write Root Map, Write Root and Chapter Maps, Write Single Map, Write Chapter Map

FmDpiVal

If set to a value greater than "0" sets @outputclass='fmdpi:*VAL*' when processing images, otherwise sets the @height and @width values based on the actual image size.

Default: `0`

Used by: Fix Images

[TopicHeadings]

Delim

Delimiter character used in H<n> lists.

Default: (space)

Used by:Topic Report

H0, H1, H2, H3, H4

"Delim"-delimited lists of paragraph tags used as topic headings in the unstructured FM files. The H0 parameter is the "chapter title", and H1, H2, H3, H4 parameters are the headings within that chapter. Assign the heading styles to these parameters for each that is to become a topic. Note that this setup is not used to actually perform the writing of XML topics, only for the Topic Report heading count and analysis.

Default: (none)

Used by:Topic Report

[AltTableTypes]

Count

Specifies the number of alternate table type entries defined in this section.

Valid values: Integer value

Default: 3 (uses internal definitions for simpletable, choicetable, and properties tables if this parameter is not defined)

Used by: Fix Tables

<N>

Each sequential numeric entry (starting with "1") defines the structure of a table that will replace the default structure (thead, tbody, row, entry). This entry defines the table type, followed by the ">" character followed by

a space-delimited table structure definition. The format for this entry is as follows:

tableType > theadReplacement theadRow theadCell(s) tbodyReplacement tbodyRow tbodyCell(s)

Where

- *tableType* is the name of the alternate table (like "simpletable"). This is followed by the ">" character which separates it from the table structure definition.
- *theadReplacement* is the name of the table head container (like "fm-simpletablehead")
- *theadRow* is the name of the row element(s) in the table head (like "sthead")
- *theadCell(s)* is the name of the cell element(s) in the table head (like "stentry"). This can specify a group of sequential cell elements, by using the following syntax: (cell1Elem,cell2Elem,...). No spaces are allowed in this group syntax.
- *tbodyReplacement* is the name of the table body container (like "fm-simpletablebody")
- *tbodyRow* is the name of the row element(s) in the table body (like "strow")
- *tbodyCell(s)* is the name of the cell element(s) in the table body (like "stentry"). This can specify a group of sequential cell elements, by using the following syntax: (cell1Elem,cell2Elem,...). No spaces are allowed in this group syntax.

Default: (none, unless Count is not specified)

Used by: Fix Tables

[MoveMarkers]

NumMarkerTypes

Number of marker types being moved. If more than 1, each marker-move specification is defined in a new INI section named MoveMarkers-*N*. This parameter is not recognized in MoveMarker-*N* sections.

Default: `1`

Used by: Move Markers

MarkerType

Name of marker type to move.

Default: `Index`

Used by: Move Markers

PrologElemPath

XPath-like specification of the placement of the marker (or content) if moved to the prolog. Specifies the path within the prolog element. If the last node specifies an element name and that element is the same as the element being moved, it will be moved to that location. If the element name differs from that being moved, then the text of the marker is copied to the specified element name.

If the last node is an attribute (as in "resource/@id"), the text of the marker is copied into that attribute value. To add a static attribute value, append a vertical bar followed by *@attr=value* (as in "|@name=TopicAlias").

Default: `metadata/keywords/fm-indexterm`

Used by: Move Markers

ParaTags

Space-delimited list of paragraph elements that are allowed to contains these markers.

Default: `p li entry`

Used by: Move Markers

MoveTopicTitleMarkersTo

Specifies the destination for markers found in topic titles. If "Prolog," markers are moved into the current topic's prolog (created if it doesn't exist). If "FirstPara," markers are moved to the first paragraph (matching a "ParaTags" entry) that follows the title. Note that this setting only applies to topic titles; other titles use the setting from the MoveParaMarkersTo parameter with the exception of BeginPara or EndPara using the next "ParaTags" element.

Valid values: Prolog, FirstPara

Default: `Prolog`

Used by: Move Markers

MoveParaMarkersTo

Specifies the destination for markers found in paragraphs (basically, anywhere other than a title). If "Prolog," markers are moved into the current topic's prolog (created if it doesn't exist). If "BeginPara," markers

are moved to the beginning of the current paragraph, or if "EndPara," markers are moved to the end of the current paragraph.

Valid values: Prolog, BeginPara, EndPara

Default: `BeginPara`

Used by: Move Markers

[MoveMarkers-*N*]

Sections that contain parameters matching those in the MoveMarkers section (described above). if the MoveMarkers/NumMarkerTypes parameter is greater than 1, there should be one MoveMarker-*N* section for each marker type being moved (starting with MoveMarkers-1 up to the value specified in the NumMarkerTypes parameter).

[CondToCharTag]

<condname>

Specifies the character tag to which the condition (<condname>) is assigned. Entering pairs of <condname>=<chartag> entries in this section defines the mapping that is performed by the Condition to Char Tag command.

Used by:Condition to Char Tag

[RenameCondMap]

<oldname>

Specifies the new condition name. Entering pairs of <oldname>=<newname> entries in this section defines the mapping that is performed by the Rename Conditions command. If no mapping is defined in this section, the Rename Conditions command does nothing.

Used by:Rename Conditions

[CondToAttrMap]

<condname>

Specifies the filtering attribute to which the condition (<condname>) is assigned. Entering pairs of <condname>=<attrname> entries in this

section defines the mapping that is performed by the Condition to Attribute command. If no mapping is defined in this section, the conditions are assigned to the attribute specified in the General/DefaultCondToAttr parameter.

Used by:Condition to Attribute

[ImagePathMap]

<origpath>

Specifies the new path to which the original path (<origpath>) is mapped. Entering pairs of <origpath>=<newpath> entries in this section defines the mapping that is performed by the Fix Images command. If no mapping is defined, the images remain referenced at the original location. The paths specified in this section must be the full path to the image file name; partial paths are not mapped. Each of the values for <origpath> and <newpath> must not be greater than 120 characters.

Used by:Fix Images

[TagCleanup]

UseTagsFromFile

Specifies where to get the list of character tags. If set to "1" the tags are read from the current file, if set to "0" the list of tags is read from the TagCleanup section (see Count and <N> below).

Valid values: Integer value (0 or 1)

Default: `1`

Used by:Tag Cleanup

NewCharColor

Specifies the color applied to newly created character tags.

Valid values: String value of a valid color name

Default: `Red`

Used by:Tag Cleanup

Count

Specifies the number of character tag entries in this section.

Valid values: Integer value

Default: `0`

Used by:Tag Cleanup

<N>

Character tag entries. One entry per tag, using sequential numeric key values starting with "1".

Default: (none)

Used by:Tag Cleanup

[RetagParas-*N*]

NumPasses

Specifies the number of iterations or passes performed by the Retag Paras command. For values greater than 1, a new RetagParas-*N* section must be added."NumPasses" is only valid in the initial RetagParas section.

Valid values: Integer value of 1 or more

Default: `1`

Used by:Retag Paras

Delim

Specifies the character that separates the tag and prefix. The only reason to change this from the default is if your paragraph tags contain the ">" character This value should be a character that does not occur in your paragraph tag names. "Delim" is only valid in the initial RetagParas section.

Valid values: Any single character

Default: >

Used by:Retag Paras

Count

Specifies the number of tag/prefix entries in this section. If you are using multiple passes (NumPasses > 1), you must provide the Count parameter in each section.

Valid values: Integer value

Default: `0`

Used by:Retag Paras

<N>

Tag/prefix entries.

Valid values: Tag and optional prefix name separated by the Delim character. One item per tag/prefix entry, using sequential numeric key values starting with "1".

Default: (none)

Used by:Retag Paras

[RetagTablesInParas]

AddPostion

Specifies that the TblFmtStr values are added as a prefix or suffix to the current table format name.

Valid values: "Prefix" or "Suffix"

Default: `Prefix`

Used by:Retag Tables in Paras

Delim

Specifies the delimiter character used in the TagList-<N> entries.

Valid values: Any single character

Default: (space)

Used by:Retag Tables in Paras

Count

Specifies the number of TagList-<N> and TagFmtStr-<N> pairs.

Valid values: Integer value

Default: `0`

Used by:Retag Tables in Paras

TagList-<N>

List of paragraph tags separated with the "Delim" character. One entry per table prefix/suffix.

Valid values: Paragraph tag names.

Default: (none)

Used by:Retag Tables in Paras

TagFmtStr-<N>

Prefix/suffix string added to the current table format name.

Valid values: String value.

Default: (none)

Used by:Retag Tables in Paras

[BuildMenucascades]

SourceType

Specifies the construction of elements to process. If you are matching on a single element that contains multiple "menu items" separated with a delimiter, the SourceType is "0". If you are matching on multiple elements separated with a delimiter, the SourceType is "1".

Valid values: Integer value (0 or 1)

Default: `0`

Used by:Build Menucascades

MatchSpec

Specifies the element to match for performing this operation. You can specify and element name or an element and attribute (b/@output-class='uicontrol').

Valid values: String value

Default: `uicontrol`

Used by:Build Menucascades

Delim

Specifies the menu delimiter to split on (typically a ">").

Valid values: String value

Default: >

Used by:Build Menucascades

WrapperElem

Specifies the element name to be the wrapper for the menucascade.

Valid values: String value

Default: `menucascade`

Used by:Build Menucascades

InnerElem

Specifies the element name within the menucascade.

Valid values: String value

Default: `uicontrol`

Used by:Build Menucascades

[Hypertext]

GotolinkHref

Defines the syntax of the resulting @href attribute. Can be used with building blocks.

Valid values: String value

Default:
`http://gotolinkurl?marker=<$URI-MARKERTEXT>&link=<$URI-LINKTEXT>`

Used by:Map Hypertext Markers

NewlinkToData

If set to "1", specifies that a "NEWLINK" marker should convert into fm-data-marker elements.

Valid values: Integer value

Default: `0`

Used by:Map Hypertext Markers

[RellinksToReltable]

ElemName

Processes all cross-ref objects in the specified element.

Valid values: String value

Default: `related-links`

Used by:Related Links to Reltable

DeleteElem

If set to "1", deletes the element specified by the ElemName parameter after processing.

Valid values: Integer value

Default: `0`

Used by:Related Links to Reltable

[AFrameToRaster]

ImageType

Specifies the type of image to be created from the anchored frame.

Valid values: JPG, PNG, GIF, TIF, or a valid export filter hint string

Default: `JPG`

Used by:AFrame to Raster

ImageFilenameTemplate

Defines the "template" for the generated graphic file names. New building blocks are available for this template: <$IMG-DOCNAME> (the document name), <$IMG-IMGNAME> (the first image file name in anchored frame [may be null]), and <$IMG-COUNT> (the nth aframe processed).

Valid values: String

Default: `image_<$IMG-COUNT>.jpg`

Used by:AFrame to Raster

ImagePath

Specifies the relative path where the generated graphics are written.

Valid values: String

Default: `graphics`

Used by:AFrame to Raster

ImageDpi

Specifies the DPI for the generated graphics.

Valid values: Integer

Default: `150`

Used by:AFrame to Raster

ConvertTypes

A space-delimited list of graphic file types (file extensions). Matching file types found while processing are converted to the new type, even if there is just one object in the anchored frame.

Valid values: String

Default: (none)

Used by:AFrame to Raster

RELATED INFORMATION:

"Topic and Map Template Building Blocks" on page 43
"Write INI for Document (or Book)" on page 52
"Edit INI for Document (or Book)" on page 52

Topic and Map Template Building Blocks

Building blocks control the file name format and structure for topics and maps.

The *fm2dita.ini* file includes two settings that require the use of file name building blocks. Most of the building blocks are the same as those provided in DITA-FMx (all except the "$FMX_" types), in addition to a four more "$FM_" types. A building block is a string of text enclosed in angle brackets and are used in the TopicNameTpl and MapNameTpl INI parameters.

The `$TITLE*` building blocks provide various options for using the text of the topic title. In all cases the text returned is processed by stripping the "stop words" as defined in the StopWords parameter. If you don't want sop words stripped, set the StopWords parameter to an empty string (nothing). The string returned by the `$TITLE*` building blocks also has punctuation deleted (other than underscores or dashes for those related building blocks). If you don't want the punctuation to be deleted, but replaced with another character, set the PunctuationChar parameter to that value. In all cases, spaces or punctuation characters will collapse into a single character.

Building Block Modifiers

You can include various types of modifiers after the building block name in square brackets to modify the resulting value.

Entering a number (from 0 to 99), limits the length of the resulting string to that value (the first *N* characters). If you want to extract a substring from a building block, include the start and end positions in square brackets. For example, the following building block will extract the first two characters from the topic type:

```
<$TOPIC_TYPE[2]>
```

Or, to extract the second through fifth characters, use the following syntax:

```
<$TOPIC_TYPE[2-5]>
```

Other modifiers can be used to change the case of the text that results from the building block. These single-character modifers must follow any numeric modifiers if present. The following modifiers are available:

- U - uppercase
- L - lowercase
- T - title case

The following syntax will generate the first two characters from the topic type in uppercase:

```
<$TOPIC_TYPE[2U]>
```

An additional "split" building block modifier is available for extracting a "field" from a delimited string. If the string being processed ($MARKERTEXT for example) contains values separated by colons, you can use this modifier to return the nth field from that string. Use the following syntax:

```
[<index><splitchar>S]
```

where <index> specifies the 1-based field and <splitchar> is the delimiter character (a single character). The following example will return the second "field" in a colon-delimited marker text string:

```
<$MARKERTEXT[2:S]>
```

Building Blocks

Valid building blocks are listed below. Some of these building blocks will make more sense for use in specific situations than others. Using an invalid building block may have unexpected results; be sure to thoroughly test on a small set of files if you're unsure.

- <$FM_VOLNUM> - volumne number (or letter) defined in the FM file
- <$FM_CHAPNUM> - chapter number (or letter) defined in the FM file
- <$FM_FILENAME> - file name (without path or extension) of the FM file
- <$FM_USER> - from *maker.ini* RegInfo/User
- <$FM_COMPANY> - from *maker.ini* RegInfo/Company
- <$OS_USERNAME> - %username% environment variable
- <$OS_COMPUTERNAME> - %computername% environment variable
- <$T_YYYY> - 4 digit year
- <$T_YY> - 2 digit year
- <$T_MM> - 2 digit month (zero padded)

- <$T_MON> - 3 character month
- <$T_MONTH> - full month name
- <$T_DD> - 2 digit date (zero padded)
- <$T_HOUR> - 2 digit hour (zero padded)
- <$TITLE> - the actual text of the title (with "stop words" removed)
- <$TITLE_NOSPACE> - the text of the title with spaces removed
- <$TITLE_NOSPACECAMEL> - the text of the title, with spaces removed, using "camel" casing
- <$TITLE_NOSPACECAMELLOW> - the text of the title, with spaces removed, using "camel" casing, and the first character lowercased
- <$TITLE_SPACETOUNDER> - the text of the title with spaces replaced with underscores
- <$TITLE_SPACETODASH> - the text of the title with spaces replaced with dashes
- <$UNIQUEID> - the unique ID as applied to the root topic element
- <$TOPIC_TYPE> - the topic type's element name
- <$VAR(*VARNAME*)> - the value of the *VARNAME* variable
- <$MARKERTEXT> - the value of the current marker's marker text (for use with the Map Hypertext Markers command)
- <$LINKTEXT> - the value of the current marker's "link text" (for use with the Map Hypertext Markers command)
- <$URI-VAR(*VARNAME*)> - the value of the *VARNAME* variable, URI-encoded
- <$URI-MARKERTEXT> - the value of the current marker's marker text, URI-encoded (for use with the Map Hypertext Markers command)
- <$URI-LINKTEXT> - the value of the current marker's "link text," URI-encoded (for use with the Map Hypertext Markers command)
- <$IMG-COUNT> - the nth anchored frame processed. (For use with the AFrame to Raster command.)
- <$IMG-DOCNAME> - the current document name containing the anchored frame being processed. (For use with the AFrame to Raster command.)
- <$IMG-IMGNAME> - the first image file name in anchored frame. Note that this may be null, if there are no referenced images in the frame. (For use with the AFrame to Raster command.)

NOTE: *Using time-based building blocks can result in errors if the conversion time spans the time unit boundary used in the building block. Use these building blocks with care.*

RELATED INFORMATION:
"Editing the fm2dita.ini File" on page 26

Programmatic Control of FM2DITA

FM2DITA provides scripting and API access to many of the commands.

In order to facilitate custom automation of your conversion process, the FM2DITA commands intended for non-interactive operation can be used with the CallClient function. These commands are assigned a special "API code" that runs that command using the current default settings (as defined in the *fm2dita.ini* file). The CallClient function is available in the FrameMaker FDK "C" programming interface as well as FrameMaker ExtendScript (release 10 and later), and FML FrameScript.

All CallClient calls use the same arguments. First is the client name, typically "Pubs-Tools:FM2DITA", but it may be something else if you have modified the entry in the APIClients section of the *maker.ini* file. The client name argument must match the associated parameter name in the *maker.ini* file. The second argument is the API code for the command. At this time you cannot pass any other arguments to the commands.

When calling an API code, the operation will be applied to the document or book that currently has the focus. Any errors or information will be written to the FrameMaker console.

The following example is the FDK code to call the **Show All Conditions** command:

```
F_ApiCallClient("Pubs-Tools:FM2DITA", "SHOWALL-CONDS")
```

The following example is the same call using ExtendScript:

```
CallClient("Pubs-Tools:FM2DITA", "SHOWALL-CONDS");
```

The following list maps the API codes to the associated commands:

- `SHOWALL-CONDS` - Show All Conditions
- `BOOK-TO-DOC` - Book to Doc
- `STRUCT-MARKER-CROSSREFS` - Struct Cross-refs to Marker Cross-refs

- RENAMEALL-CONDS - Rename Conditions
- FIX-CROSSREF-FMTS - Fix Cross-ref Formats
- TABLE-TO-TEXT - Table to Text
- RETAG-PARAS [<groupName>] - Retag Paras. Accepts optional <groupName> parameter if multiple groups are defined in INI file. Separate the command name from the parameter with a verical bar ("|") or "\t" character sequence. For example, use the following syntax to run the group "firstpass" using ExtendScript:

```
CallClient("Pubs-Tools:FM2DITA", "RETAG-PARAS|firstpass");
```

- RETAG-TABLES-PARAS - Retag Tables in Paras
- UNTAG-SPACES - Untag Boundary Spaces
- DELETE-EXTRA-CROSSREFS - Delete Extra Cross-ref Markers
- FLATTEN-CROSSREF-FORMATS - Flatten Cross-ref Formats
- AFRAMETORASTER - AFrame to Raster
- COND-TO-CHAR - Condition to Char Tag
- IMPORT-TPL-EDD - Import Template and EDD
- ASSIGN-IDS - Assign IDs to Topics
- UNWRAP-ELEMS - Unwrap Elements
- DELETE-ELEMS - Delete Elements
- COND-TO-ATTR - Condition to Attribute
- FIX-IMAGES - Fix Images
- FIX-TABLES - Fix Tables
- FIX-XREFS - Fix/Flatten Cross-refs
- MAP-HYPERTEXT - Map Hypertext Markers
- RELLINKS-TO-RELTABLE - Related Links to Reltable
- FLATTEN-CROSS-REFS - Flatten Cross-refs
- MOVE-MARKERS - Move Markers
- VAR-TO-CONREF - Variables to Conrefs
- MENUCASCADES - Build Menucascades
- MERGE-TAGS - Merge Code Lines
- TAB-TO-SPACE - Tab to Spaces

- `DELETE-ATTRS` - Delete Invalid Attributes
- `DELETE-MARKERS` - Delete Unstructured Markers
- `DELETE-EMPTY-ELEMS` - Delete Empty Elements
- `WRITE-ROOT-MAP` - Write Root Map
- `WRITE-ROOT-CHAP-MAPS` - Write Root and Chapter Maps
- `WRITE-SINGLE-MAP` - Write Single Map
- `WRITE-CHAPTER-MAP` - Write Chapter Map
- `WRITE-TOPICS` - Write XML Topics

RELATED INFORMATION:

"FM2DITA Commands" on page 49

2 FM2DITA Commands

Commands perform various operations of the conversion process.

Most of the FM2DITA commands can be run on a book or file. If a command is only available for one or the other, that is indicated in the title.

These commands appear in the FM2DITA menu in the general order they should be run. Not all commands are needed for all conversions, but in most cases you should not run a command listed later on the menu before one listed higher on the menu.

When run on a book file, if a referenced document is not open, it is opened, processed and saved; referenced documents that are open, are processed but not saved. In general, it is best to run these commands on documents that are open, so you can review any errors before saving the document.

RELATED INFORMATION:
"Using FM2DITA" on page 1
"Editing the fm2dita.ini File" on page 26
"Programmatic Control of FM2DITA" on page 46
"Preconversion Pod" on page 50
"Conversion Pod" on page 50
"Write XML Pod" on page 50

Reports and Command Control

EDD Element Browser (file)

Displays an alphabetic listing of element definitions in the current EDD

This command is helpful during EDD development and updates. Double-click an element name in the list to quickly scroll to that element definition.

This command can only be used on an EDD file.

RELATED INFORMATION:
"FM2DITA Commands" on page 49

Preconversion Pod

Provides easy access to all of the preconversion commands.

Use this modeless dialog for quick access to the preconversion commands.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Conversion Pod" on page 50
"Write XML Pod" on page 50

Conversion Pod

Provides easy access to all of the conversion commands.

Use this modeless dialog for quick access to the conversion commands.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Preconversion Pod" on page 50
"Write XML Pod" on page 50

Write XML Pod

Provides easy access to all of the commands for writing new XML files.

Use this modeless dialog for quick access to the commands that write new XML files.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Preconversion Pod" on page 50
"Conversion Pod" on page 50

Topic Report

Reports on the number of topics in the current file or book.

Before this command can be used, you must set up the TopicHeadings section of the *fm2dita.ini* file. Review the TopicHeadings documentation in Editing the fm2dita.ini File.

This command is useful for estimating the number of topics in a book based on the specification of different heading levels.

RELATED INFORMATION:
"Editing the fm2dita.ini File" on page 26

Catalog Report

Generates a report on the actual use of various styles, formats, and object types in the current file or book.

The following items are included:

- Paragraph styles
- Character styles
- Cross-ref formats
- Variable definitions
- Table formats
- Marker types

Only the object definitions actually used in one or more files are included in the report. This is useful for conversion table development and general planning.

RELATED INFORMATION:
"Condition Report" on page 51

Condition Report

Scans the current file or book and reports the conditions in use and possible inline condition usage.

This command provides a quick way to get the list of all visible conditions in use in a file or book. Conditions that are hidden or not applied to content are not included in the report. The condition names are displayed in the FrameMaker console window.

If any conditional ranges are found that don't start at the beginning of a paragraph, these are reported separately. Typically, inline conditions do not convert properly, so you may want to locate these text ranges and update the conditional tagging.

RELATED INFORMATION:

"Rename Conditions" on page 53
"Show All Conditions" on page 52
"Condition to Attribute" on page 67

Write INI for Document (or Book)

Creates a new INI file in the current document's or book's folder.

The *fm2dita.ini* file must exist in the folder that contains the files being processed. This command provides an easy way to create the default *fm2dita.ini* file.

RELATED INFORMATION:

"Edit INI for Document (or Book)" on page 52
"Editing the fm2dita.ini File" on page 26

Edit INI for Document (or Book)

Opens the current document's or book's INI file for editing.

To modify the operation of FM2DITA commands requires editing of the *fm2dita.ini* file that exists in the folder that contains the current document or book. This command opens that INI file using the associated editor. If no *fm2dita.ini* file is found, you are prompted to create a new file.

RELATED INFORMATION:

"Write INI for Document (or Book)" on page 52
"Editing the fm2dita.ini File" on page 26

Show All Conditions

Shows all conditions in the current file or book.

It is always a good idea to show all conditions in all files as preparation to applying a conversion table.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Condition Report" on page 51
"Rename Conditions" on page 53
"Condition to Attribute" on page 67

Preconversion Tools

Utilities for preparing files for conversion table processing.

RELATED INFORMATION:
"FM2DITA Commands" on page 49

Book to Doc

Creates a single FM file from all files in a book.

This command is useful when developing a conversion table. Because the conversion table generation command reads from a single file, use this command to make a single file from all files in a book to ensure that the generated conversion table represents all styles and objects.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Struct Cross-refs to Marker Cross-refs

Relinks structure-based cross-references to marker-based cross-references.

The FM2DITA tools are designed to process unstructured content to DITA. If you are starting with structured FM files, you will likely need to strip the structure from the files and continue by using the object to element based mapping provided by a conversion table.

However, before stripping the structure, you should run this command to transfer the cross-ref linking from the structured model to markers.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Rename Conditions

Renames conditions in the current file or book.

This command requires setup to the RenameCondMap section of the *fm2dita.ini* file. Once this INI section is updated with oldname/newname pairs, run it on a file or book to rename the conditions in all files.

The Rename Conditions command should be used to clean up condition naming before using the Condition to Attribute command. It is always best to use the Show All Conditions command before using this command.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Condition Report" on page 51
"Show All Conditions" on page 52
"Condition to Attribute" on page 67

Fix Cross-ref Formats

Strips leading/trailing spaces from Cross-ref format names.

Cross-ref format names that contain leading or trailing spaces can cause problems down the road in your conversion. If your cross-ref format names may have leading or trailing spaces, run this to clean things up.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Tag Cleanup

Provides a method for reviewing character tagging and applying new styles.

The Tag Cleanup dialog displays a list of character tag names. Choosing the Next button selects the next instance of bold or italic text in the document. If the selected text range is tagged with a character style, that name displays in the bottom of the dialog, if no tag is applied, the font property displays as “<italic>” or “<bold>”. To change the tagging for this range, select a tag name from the list, and choose Apply Tag, or click the Remove Styling button. Choose the Next button to move to the next instance of inline formatting.

By default, the tags listed in the dialog are the tag names defined in the document. If you’d like to use a subset of the tags, or assign tags that do not exist in the document, you’ll need to edit the TagCleanup section of the *fm2dita.ini* file. The following example shows how to specify that only 4 tags are available in this dialog.

```
[TagCleanup]
UseTagsFromFile=0
NewCharColor=Red
Count=4
1=uicontrol
2=varname
```

```
3=userinput
4=wintitle
```

Use this command any time before applying the conversion table.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Tables to Text

Converts all tables of the specified format(s) to text.

It may be necessary to convert some tables to text before applying the conversion table. This command prompts for the table format(s) to convert. If multiple formats need to be converted enter each format name separated with a space. The tables are converted to text, row by row and cell by cell, working from top to bottom and left to right. Each paragraph in the table cells become a new paragraph.

If the format names are added to the UnwrapTables parameter in the *fm2dita.ini* file, you will be prompted with a confirmation dialog rather than a text entry dialog. The UnwrapTablesPrefix parameter can be used to add a prefix to the paragraphs that result from the conversion process. Each of these INI parameters are space-delimited lists. If both lists have the same number of entries, the prefixes are applied to the content from the corresponding table. If the UnwrapTablesPrefix list doesn't provide a corresponding entry, the first entry in that list will be used.

Refer to the information in Editing the fm2dita.ini File for additional information on these INI parameters.

Run this command before applying a conversion table.

Complex mapping option

If you have tables that require a variable or detailed mapping configuration, an expanded syntax may be used in the UnwrapTablesPrefix parameter. This starts by using a vertical bar ("|") as the delimiter, then bracketed rules within each delimited region to specify a column number and a paragraph tag to match. When these rules match the condition in the table, a prefix can be assigned to the new paragraph tag or a new tag may be assigned.

The following describes the parameters of the syntax for this method:

`UnwrapTablesPrefix=|{`*`colnum`*`:`*`matchtag`*`>`*`prefixtag`*`}|`

Where:

colnum

The column number (starting with "1") or asterisk ("*") for any column.

matchtag

The paragraph tag to match or ("*") for any tag.

prefixtag

The prefix to assign to the unwrapped table cell paragraph tag. If you want to assign a new tag name use "=" as the first character of the *prefixtag* parameter.

For example, use the following rules if you have a 2-column table, where the first column should always be converted to a paragraph tag named "icon", and the second column tag name should be prefixed with "info-".

```
UnwrapTablesPrefix=|{1:*>=icon}{2:*>info-}|
```

Note the use of the "=" to assign a tag name, and in both cases, this is mapped regardless of the paragraph tag name in each column.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Retag Paras

Renames paragraph tags based on surrounding paragraph tags.

If you are converting into multiple structural models, it is typically required to have unique paragraph tags in each model group. This is needed to differentiate between concept and task, for example, but also within topics (like task) to identify paragraphs within different sections. This command assigns a prefix to specified paragraph tags based on tag matching rules defined in the RetagParas section of the *fm2dita.ini* file.

This command can make multiple passes over a document, allowing you to build up "layers" of tag names based on changes made in a previous pass. There are two basic types of rules that can be used. A "tag until" rule starts assigning a prefix to the tag of all paragraphs that follow a specified paragraph tag until it reaches another specified paragraph tag. A "look ahead" rule assigns a prefix to a specified paragraph's tag if the paragraph tags that follow match the specified rule. Using a combination of these rules provides very powerful retagging capabilities.

This command requires initial setup in the RetagParas section of the *fm2dita.ini* file. The following options may be set:

- NumPasses - the number of passes or iterations to perform. For more than one pass, you'll need to add additional RetagParas-<N> sections to the INI file. Default value: "1"
- Delim - the delimiter character used to separate the tag name from the associated prefix (described below). Default value: ">"
- Count - the number of tag/prefix rules in this section.
- <N> - tag/prefix rules identified by sequential numeric values (starting with "1"). Depending on the format of the tag/prefix rules, different types of processing is performed (see below for details and examples). The prefix is optional, and if provided is separated from the tag name by the "Delim" character.

Refer to the information in Editing the fm2dita.ini File for additional information on these INI parameters.

This command must be run before applying the conversion table (since the conversion table should be designed to key off of these new tag names).

Tag/Prefix rule syntax

Although there are two basic types of rules, "tag until" and "look ahead", there are three formats for rules that can be used in a RetagParas section. The fundamental syntax for all rules is as follows (note that the square brackets here are actual characters, not indicators of optional content):

TAG[*MATCHRULE*]>*PREFIX*

Where:

TAG

Specifies the paragraph tag to operate on or from (depending on the existence of the *MATCHRULE*). All *TAG* values must be unique in each section; if you need to use different *MATCHRULE* values for the same *TAG* name, create additional sections and increment the number of NumPasses.

MATCHRULE

Only used for the "look ahead" type of retagging rule. Defines the sequence of paragraph tags that may follow the *TAG* paragraph. The syntax of the *MATCHRULE* loosely follows the syntax used for DTD general rule specification. The MATCHRULE can use parenthesis to group tag names separated with a vertical bar, but nested groups are not supported. The following operators are supported to indicate the frequency or existence of tags: "*" (zero or more), "+" (one or more), and "?" (optional). These operators must never be used within a group of tag

names, and should follow the closing parenthesis if used. If no operator is used, that tag is required in the sequence.

PREFIX

A prefix string that is applied to the matching paragraph tag names. Depending on the existence of a *MATCHRULE*, the prefix string is either applied to the *TAG* paragraph or the paragraph tags that follow the *TAG* paragraph. The *PREFIX* is separated from the *TAG*/*MATCHRULE* by the Delim character (default ">") indicated in the INI file. The *PREFIX* is optional.

If the *PREFIX* value starts with a "=", the value is assigned as the tag name rather than used as a prefix.

If the *TAG* is specified with a following *PREFIX*, the command applies that prefix string to all paragraph tags that follow each *TAG* paragraph tag, until another *TAG* paragraph tag is encountered.

If the *TAG* is specified with no *PREFIX*, it tells the command to stop applying the prefix without adding a new prefix to later paragraph tags.

If the *TAG* is specified with a following square bracketed *MATCHRULE* descriptor, that descriptor phrase tells the command to only match a *TAG* when the sequence of tags indicated in the *MATCHRULE* is a match. When this match occurs, the *PREFIX* specified (required in this case) is applied to the *TAG* paragraph tag, and not the tags that follow.

"Tag Until" rules

A document may use H0, H1, and H2 tags for the general headings and a ToDo tag for procedures. The following INI section will retag the paragraphs in the H1 and H2 sections with a "con-" prefix, and the paragraphs in the ToDo sections with a "task-" prefix. It leaves the paragraphs in the H0 section without a prefix (no delimiter character or prefix is specified for the H0 tag).

```
[RetagParas]
NumPasses=1
Delim=>
Count=4
1=H0
2=H1>con-
3=H2>con-
4=ToDo>task-
```

If you want to apply multiple levels of prefixes (often useful to differentiate sections in task topics), set the NumPasses parameter to the number of passes to perform. When applying multiple passes on a document, you should start with the "lowest" level of tags. The following INI sections provide for separate

tagging of the "step" content from any content that might precede the steps (in the "context" section of a task).

```
[RetagParas]
NumPasses=2
Delim=>
Count=5
1=H0
2=H1
3=H2
4=ToDo
5=step>step-

[RetagParas-2]
Count=4
1=H0
2=H1>con-
3=H2>con-
4=ToDo>task-
```

You'll notice that in the first pass (RetagParas section), only the "step" tags are "prefix restart" paragraphs. The other tags are included so the retagging will stop when the task ends. In the second pass (RetagParas-2 section) an additional prefix is applied, and in this section the "step" tag is omitted so it will itself get another prefix, along with any paragraphs in each step.

"Look Ahead" rules

The look ahead rules make use of the *MATCHRULE* descriptor phrase that follows the *TAG* in a rule definition. This is used when your source document doesn't contain special heading paragraph tags that you can use to identify specific topic groups. For example, if a document doesn't use a ToDo tag for task headings, but always starts a task with some Body paragraphs followed by a numbered list with the tag NumStep. This may happen after a H1, H2, or H3 heading.

This type of rule is typically used with a following "tag until" rule in a multipass operation. In the example below, the first pass applies a "task-" prefix to the H1, H2, and H3 headings that match the *MATCHRULE* descriptor. It leaves all other tags alone. In the second pass, all unprefixed headings are assumed to be concept topics and apply the "con-" prefix to all content after each heading. It also assigned the "task-" prefix to all content after the task-tagged headings.

```
[RetagParas]
NumPasses=2
Delim=>
Count=3
1=H1[Body*,NumStep]>task-
2=H2[Body*,NumStep]>task-
3=H3[Body*,NumStep]>task-
```

```
[RetagParas-2]
Count=7
1=H0
2=H1>con-
3=H2>con-
4=H3>con-
5=task-H1>task-
6=task-H2>task-
7=task-H3>task-
```

RetagParas Groups

You can create multiple groups of rules for the RetagParas command. To make use of this feature, add a RetagParaGroups section to the INI file.

```
[RetagParasGroups]
Count=2
1=First Group
2=Second Group
```

With this set up, when you run the RetagParas command, you'll get a dialog prompting you to select the group to run.

When using this feature, the section names for the first group will start with "RetagParas1", the second group will start with "RetagParas2", and so on. For example, what would normally be the main RetagParas section will be RetagParas1. If this group has multiple passes, what would normally be "RetagParas-2" is now "RetagParas1-2".

All of the functionality within a group is identical to before.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Retag Tables in Paras

Applies new table format names based on the associated paragraph tag.

In order to control proper nesting of tables in complex structured, it is at times necessary to assign new table format names to tables within certain section of a topic. This command will retag tables by assigning a prefix or suffix to the current table format name.

Before using this command you must modify the RetagTablesInParas section of the *fm2dita.ini* file to specify the paragraph tag and prefix/suffix string mapping. Set the value of "Prefix" or "Suffix" to the AddPostion parameter to indicate where the string is added to the table format names. Set the Count parameter to the number of TagList and TblFmtStr pairs you are specifying.

Then set the TagList-<N> values to a list of paragraph tag names (delimited with the Delim character; space by default) which will be scanned for table anchors. The associated TblFmtStr-<N> value is the string that is added (as a prefix or suffix) to the table format name.

The following example shows how you would assign a prefix of "list-" to the table format names of tables that are anchored to Bullet1 or BulletCont paragraphs, and a prefix of "info-" to the table format names of tables that are anchored to Step or StepInfo paragraphs.

```
[RetagTablesInParas]
AddPosition=Prefix
Delim=
Count=2
TagList-1=Bullet1 BulletCont
TblFmtStr-1=list-
TagList-2=Step StepInfo
TblFmtStr-2=info-
```

This command must be run before applying the conversion table (since the conversion table should be designed to key off of these new tag names) and typically after running the RetagParas command.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Retag Paras" on page 56
"Editing the fm2dita.ini File" on page 26

Untag Boundary Spaces

Ensures that leading and trailing spaces are unformatted.

When character tags are mapped to elements, if the character range contains a leading or trailing space the resulting element will have a leading or trailing space. Because of XML whitespace normalization rules, any leading or trailing whitespace in an element may be deleted when saved to XML.

If, after running a full conversion to XML, and you see missing space to one side or the other of inline tagging, this is likely the problem.

This command scans for these "boundary" conditions (locations where inline tagging changes from one tag to another, or changes to no tag). Any space that exists to one side or the other of this boundary location is "untagged" (assigned "Default Para Font").

This command must be run before applying the conversion table.

RELATED INFORMATION:

"FM2DITA Commands" on page 49

Delete Extra Cross-Ref Markers

Scans for paragraphs that contain multiple Cross-Ref markers and deletes all but one.

When FrameMaker generates a structured document from unstructured content, it assigns an Id attribute to any paragraph that contains a Cross-Ref marker. The value of the Id attribute is based on the unique ID of that marker. If there are multiple Cross-Ref markers in a paragraph, it chooses one and ignores the others. If a cross-reference was linking to one of the markers that isn't used to map the Id attribute, the resulting xref will be broken.

To resolve this issue, this command scans the document for paragraphs that contain multiple Cross-Ref markers. When this condition is found, it deletes all but one of these markers from each paragraph. Running this command will likely cause cross-references to become unresolved, so it is important to re-run the Update Book command after running this command, and resolve all unresolved references.

This command must be run before applying the conversion table.

RELATED INFORMATION:
"FM2DITA Commands" on page 49

Flatten Cross-ref Formats

Sets the value of all cross-ref formats to null (an empty string) in the current file or book.

Running this command prevents the save to XML from pushing the content of a cross-ref outside of the xref element. This is not needed if you use the Fix/Flatten Cross-Refs command, but if you're not using that, you should probably run this command instead.

Note that after running this command, all resolved cross-refs will be empty (and appear to be missing), but they will convert to empty xref elements after applying the conversion table.

Run this command before applying a conversion table.

RELATED INFORMATION:
"FM2DITA Commands" on page 49

AFrame to Raster

Generates raster images from anchored frames that contain multiple graphic objects.

When referencing images, the DITA model can only reference a single file; it cannot reference a collection of images or graphic objects with callouts. If your source FrameMaker files make use of anchored frames that contain multiple referenced graphics or include graphic overlay objects (callouts, arrows, boxes, etc.), you'll need to replace those with single referenced images.

The best way to do this is to manually go through all such cases and clean them up. A common way of handling images with overlaid callouts is to replace the text callouts with numeric or character-based identifier bubbles, then add a textual legend below the graphic. This can take time, and if you don't have the time to deal with it properly, you can use this command to generate raster images from the content in the anchored frames.

This command requires initial setup in the AFrameToRaster section of the *fm2dita.ini* file. The following options may be set:

- ImageType - the graphic format generated from the objects in the anchored frame. Possible types are: JPG, PNG, GIF, TIF, or a valid export filter hint string (use the List Import/Export Filters command to get the valid export filter hint strings for your installation).
- ImageFilenameTemplate - a string that defines the "template" to be used for the graphic file names when generating the images. This string is a combination of plain text and building blocks that make up a file name. There are three special building blocks for use with this command: <$IMG-DOCNAME> (the document name), <$IMG-IMGNAME> (the first image file name in anchored frame [may be null]), and <$IMG-COUNT> (the nth aframe processed)
- ImagePath - the folder or path where the image files are generated.
- ImageDpi - the DPI of the raster images generated.
- ConvertTypes - an optional parameter that can be used to generate new raster images from existing images of specified types. This parameter is a space-delimited list of file types.

Refer to the information in Editing the fm2dita.ini File for additional information on these INI parameters.

This command must be run before applying the conversion table.

RELATED INFORMATION:
"FM2DITA Commands" on page 49

List Import/Export Filters

Generates a list of all import and export filters currently available.

The ImageType parameter used by the AFrame to Raster command accepts the following string values: JPG, PNG, GIF, TIF, or a valid export filter hint string. The valid export hint strings may vary depending on your installation. This command generates the list of all import and export filters and their associated "hint strings."

RELATED INFORMATION:

"FM2DITA Commands" on page 49

Condition to Char Tag

Assigns character tags to content based on conditional tagging.

This command requires setup to the CondToCharTag section of the *fm2dita.ini* file. Once this INI section is updated with condname/chartag pairs, run it on a file or book to replace conditional ranges with the specified character tags in all files.

The Rename Conditions command could be used to clean up condition naming before using the this command. It is always best to use the Show All Conditions command before using this command.

RELATED INFORMATION:

"FM2DITA Commands" on page 49

Import Template and EDD

Imports the specified template and EDD and applies the specified attribute display setting into the current file or book.

Before using this command, the TopicAppName parameter must be set in the *fm2dita.ini* file. The template file associated with that structure application is used as the template and EDD source.

After the template and EDD are imported, the attribute display setting is applied. The available settings correspond to those listed in the Attribute Display Options dialog found by right-clicking in the Structure View window. The setting for this command is defined in the *fm2dita.ini* file by the Attribute-Display parameter. Valid values are as follows:

- None
- ReqSpec
- All

By default this command applies the “ReqSpec” (required and specified) value.

This command must be run after the conversion table has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Check for Topic Collisions

Provides a pre-write-to-XML scan of the current file or book to ensure that the specified topic file naming scheme doesn’t result in any duplicate file names.

Based on the settings in the *fm2dita.ini* file that affect file naming, this command reports the number of files that would result in duplication or over-writing of files. It is important to run this before writing maps or topics, and should be run before proceeding with a conversion. If file naming is based on topic IDs, you will need to run the Assign IDs to Topics command before running this command. It is important to make sure that chapter numbering properties are set properly in the book or file being processed if the file naming relies on chapter numbering.

If file name collisions are reported, modify the TopicNameTpl INI parameter or edit the topic titles (if file names are based on topic titles). To ensure that file names do not collide, include the $UNIQUEID building block in the topic name template. Also, including the $FM_CHAPNUM building block can help to reduce the likelihood of collisions (just remember to update the chapter numbering in the book after importing the template and EDD).

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Assign IDs to Topics" on page 66
"Fix Cross-refs" on page 72

Assign IDs to Topics

Assigns unique ID attribute values to all topics in the current file or book.

The format of ID values assigned to topics are defined in the *fm2dita.ini* file by the IdPrefix and IdType parameters. By default IdPrefix is set to "id" and IdType is "GUID".

This command should be run before running the other FM2DITA commands to ensure all topics are properly marked. It must be run after the EDD has been applied since it relies on the default class attribute values to identify topic elements (they contain the string "topic/topic").

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Unwrap Elements

Unwraps elements of the specified type in the current file or book.

This command prompts for an element name to unwrap. It scans for all elements of the specified name and unwraps each, promoting any child elements. If multiple element names need to be processed, enter multiple element names separated by spaces.

If a table object element is specified in this command, the "Tables to Text" processing is performed and the table object, and related structure, is unwrapped. This results in the content of each table cell being promoted to be a child of the element that is the parent of the specified table object element.

You can pre-set the element names in the UnwrapElems parameter in the *fm2dita.ini* file. Refer to the information in Editing the fm2dita.ini File for details on this INI parameter.

This command should be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Delete Elements" on page 67

Delete Elements

Deletes elements of the specified type in the current file or book.

This command prompts for an element name to delete. It scans for all elements of the specified name and deletes each, including any child elements. If multiple element names need to be deleted, enter multiple element names separated by spaces.

You can pre-set these values in the DeleteElems parameter in the *fm2dita.ini* file. Refer to the information in Editing the fm2dita.ini File for details on this INI parameter.

This command should be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Unwrap Elements" on page 66
"Delete Empty Elements" on page 79

Condition to Attribute

Applies filtering attribute values based on conditional tagging in the current file or book.

This command requires initial setup in the DefaultCondToAttr parameter and the CondToAttrMap section of the *fm2dita.ini* file. The CondToAttrMap INI section allows you to define a mapping of condition name and attribute name. The DefaultCondToAttr parameter specifies the default attribute name to use for conditions where no mapping is expressly defined in the CondToAttrMap section.

IMPORTANT: *Only block-level conditional tagging is mapped to attributes, and only the tagging that "touches" the start of the block-level element is used. Also, the results from this command should be carefully reviewed for usefulness. There are situations where multiple elements will be tagged with the same attribute value because they all "touched" the same condition, but this may not be as desired. For example, if the first list item is conditionalized, the <p> tag, the <li> tag, and the <ul> tag may all receive the conditional attribute.*

For “container” elements, the conditional tagging is checked just inside the open tag. For tables (i.e., tgroup element), the conditional tagging is checked just before the table anchor.

This command must be run after the EDD has been applied and it may be useful to run this command before commands that may rearrange elements (like Fix Images or Fix Tables).

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Condition Report" on page 51
"Show All Conditions" on page 52
"Rename Conditions" on page 53

Fix Images

Performs various cleanup and adjustments to all images and container fig elements.

This command performs the following actions on each anchored frame in the document:

- Check for multiple objects in a frame, if more than one an error is reported. Processing continues using the first referenced image.
- Sets the image/@href attribute to a relative path based on the actual referenced image.
- If a path mapping is defined in the ImagePathMap section of the *fm2dita.ini* file, the @href value is modified accordingly.
- Sets the image/@height and image/@width values based on the current image scaling.
- If this image is wrapped in a fig element (or an element with a @class of “topic/fig”), the following processing is performed:
  - Set image/@placement to “break”.
  - If the alignment is center or right, set image/@align to that value.
  - Check for an element that wraps the image (between the fig and the image) with an @outputclass of “figimage”, if found, unwrap that element.

– Check for a child element of the fig with an @outputclass of “figtitle”, if found move it to the proper location (first child of fig) and retag as a title element (“figtitle” value is removed).

This command must be run after the EDD has been applied and after the Assign IDs to Topics command has been run.

The following images show a simple structure before and after running the Fix Images command.

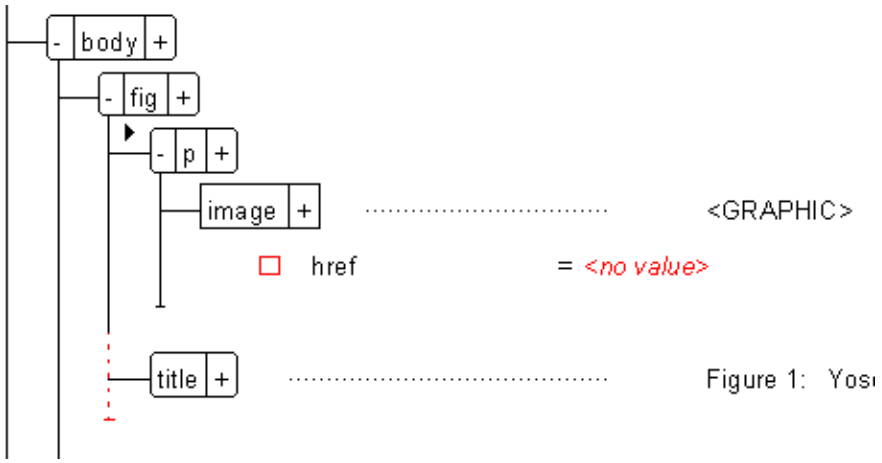

Figure 2-1: Before running the Fix Images command

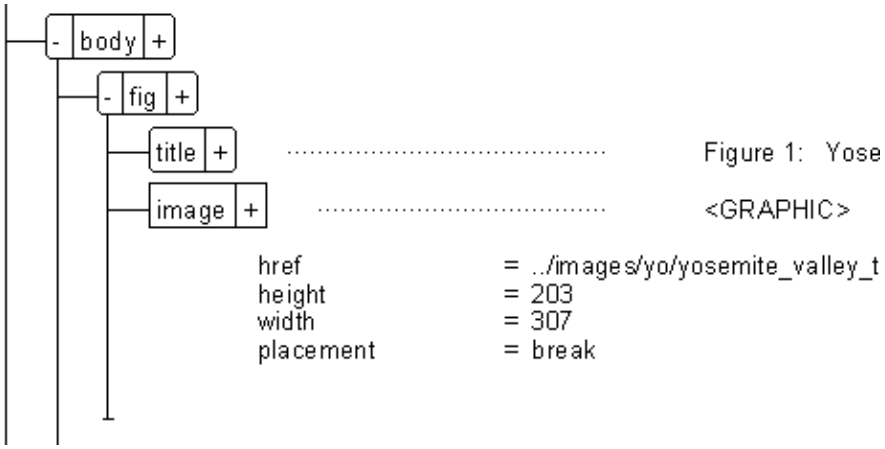

Figure 2-2: After running the Fix Images command

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Assign IDs to Topics" on page 66

Fix Tables

Performs various cleanup tasks on tables in the current file or book.

This command performs the following actions on each table in the document:

- Moves all table titles into the proper location for a DITA table. (FM tables will have the title inside of the <tgroup> element, but DITA should have them as a child of table.)
- Moves any table footers to the last row(s) of the table body. Sets the @outputclass of those rows to "tablefooter".
- If the General/MoveTblOutputclass parameter is set to 1 (the default is 0), moves the tgroup/@outputclass attribute to the parent table element.
- If the General/SaveTableWidth parameter is set to 1 (the default is 0), saves the table width as a percentage rounded to the nearest higher "5" to the parent element's pgwide attribute (if the parent element has a class of "topic/table").
- If the General/SaveTableShading parameter is set to 1 (the default is 0), cell shading information is saved to the cell element's outputclass attribute.
- If the General/TableFmtPrefixChopChar parameter is set to a character (the default is nothing), the table's format tag is truncated at that character.
- If the element associated with the table object (typically <tgroup>) is named "simpletable", the inner structural elements of the table will be renamed to match the DITA simpletable model. The <thead> element becomes <fm-simpletablehead> and <tbody> becomes <fm-simpletablebody>. The head row(s) become <sthead>, the body row(s) become <strow>, and the cells become <stentry>.

  Note that the <fm-simpletablehead> and <fm-simpletablebody> elements are not proper DITA elements but are needed within FrameMaker. The read/write rules file should unwrap these elements, and the EDD should identify them as the proper object type. The FM2DITA structured application is set up to handle this properly, but if you use another app, you'll need to make sure it is as well.

  Similar mapping is performed for <choicetable> and <properties> table elements. This mapping is performed based on the entries in the AltTableTypes section. If this section is omitted, the default processing is done. If you need to perform custom mappings, you can modify the entries in this section.

IMPORTANT: *In order for this command to properly move the table titles, your conversion table should map "TT:" to the "title" element. Also, it's best to map your "table-title" paragraph tag(s) (whatever they are called in your document) to the "p" element. After running the conversion table, you'll have the structure table/tgroup/title/p. This command will unwrap the "p" but will relocate any "id" on the "p" to the table so any references to that table should continue to link properly.*

This command must be run after the EDD has been applied and after the Assign IDs to Topics command has been run.

The following images show the processing of a simpletable, before and after running the Fix Tables command.

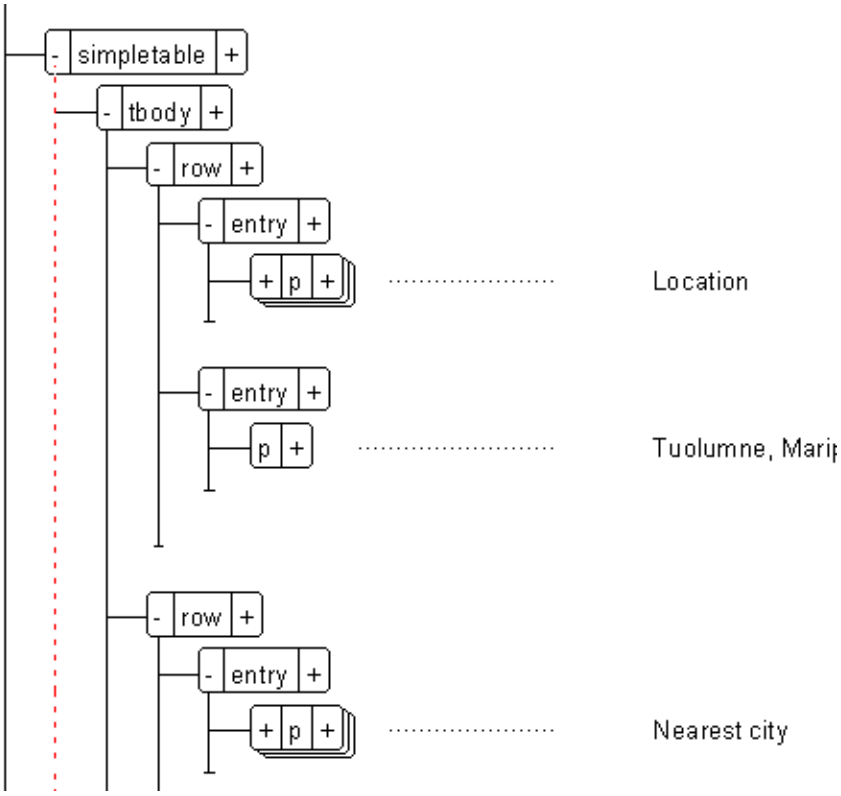

Figure 2-3: Before running the Fix Tables command

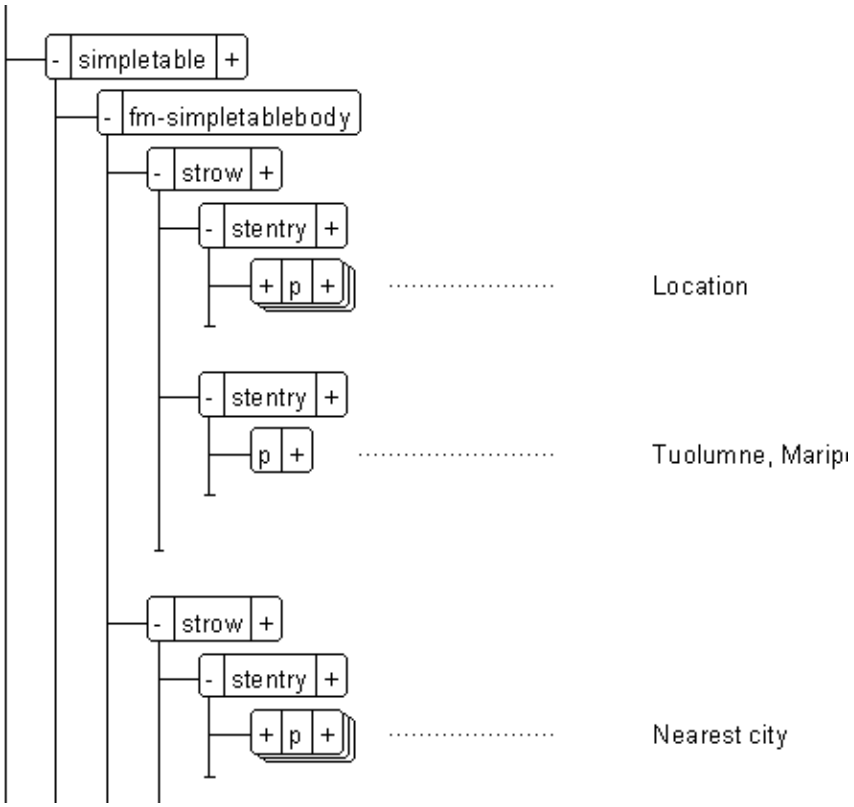

Figure 2-4: After running the Fix Tables command

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Assign IDs to Topics" on page 66

Fix Cross-refs

Relinks xrefs to the proper targets once the chapter FM files are broken into topics in the current file or book.

This command relies on proper setup of various parameters in the *fm2dita.ini* file. In particular the following parameters must be set up as needed:

- IdPrefix
- IdType
- TopicNameTpl
- FmXrefAttr
- StopWords

IMPORTANT: *Do not change the value of these INI parameters after running this command. Changes will result in misnamed files when using the Write XML Topics command as well as the map building commands.*

It is also important to set the chapter numbering in the book file if the file naming is based on chapter numbering.

NOTE: *This command only "fixes" cross-refs that are linked to unstructured objects (Cross-Ref markers or Paragraphs). If you have cross-refs that are linked to elements, they will not be processed properly.*

This command must be run after the EDD has been applied and the Assign IDs to Topics command has been run.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Topic and Map Template Building Blocks" on page 43
"Assign IDs to Topics" on page 66
"Write Root Map (book)" on page 80
"Write Root and Chapter Maps (book)" on page 81
"Write Single Map (book)" on page 81
"Write Chapter Map (file)" on page 82
"Write XML Topics" on page 83

Map Hypertext Markers

Converts the content of Hypertext markers into usable elements in the current file or book.

This command relies on proper setup of parameters in Hypertext section of the *fm2dita.ini* file. If you are converting "gotolink" Hypertext markers, set up the GotolinkHref parameter with the required format. If you are converting "newlink" Hypertext markers and want them converted into fm-data-markers, set the NewlinkToData parameter to "1". If you are converting "message URL" Hypertext markers, those will convert into @href attributes in external xref elements by default; no setup is necessary.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Related Links to Reltable

Generates a "reltable" map from the related links in a file or book.

This command relies on proper setup of parameters in the RellinksToReltable section of the *fm2dita.ini* file. It creates a 2-column reltable for each document. The first column is assigned relcolspec/@linking='sourceonly' and the second column is assigned relcolspec/@linking='targetonly'.

The ElemName parameter in the RellinksToReltable section specifies the "related-links" element. This element is scanned for descendant elements that contain @href or @xtrf attributes with a non-null value. These attribute values are assumed to be the target topic and are used as the basis for generating links in the reltable. If you'd like this group of elements deleted after processing (from the ElemName element), set DeleteElem to "1".

After processing, one or more separate maps are created that contain the relationship tables. You must manually link these "reltable" maps (with a mapref element) into your root or chapter maps, or copy/paste the reltables into those maps.

This command must be run after the Fix Cross-refs command has been run, and before Write XML Topics.

RELATED INFORMATION:
"FM2DITA Commands" on page 49
"Fix Cross-refs" on page 72
"Editing the fm2dita.ini File" on page 26

Flatten Cross-refs

Removes link text from cross-ref objects in the current file or book in preparation for export to XML.

When saving to XML, the content of a cross-ref object is "popped out" of the container element. This is not useful and results in duplication of the link text (the xref resolves and pulls in the target text, and the "popped" text follows the xref element. This command prevents this from happening.

Note that after running this command you will not see any cross-ref text in your file. This is as expected. You will see the xref elements, and they will properly resolve after export to XML.

If you are not using the Related Links to Reltable command and would prefer that the Fix Cross-refs command did the flattening as well, set the General/FlattenXrefs parameter to “1”.

This command must be run after the Fix Cross-refs command has been run.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Fix Cross-refs" on page 72
"Editing the fm2dita.ini File" on page 26

Move Markers

Moves markers (typically Index) to the start or end of the paragraph as well as out of titles in the current file or book.

This command requires initial setup in the MoveMarkers section of the *fm2dita.ini* file. The following options must be set:

- MarkerType - marker type to process (typically set to “Index”)
- ParaTags - list of paragraph elements that are valid for markers of this type
- MoveTopicTitleMarkersTo - indicates how markers found in topic titles are handled
- MoveParaMarkersTo - indicates how markers found in paragraph are handled

Refer to the information in Editing the fm2dita.ini File for details on these INI parameters.

It is important to check the console window for errors, as there are conditions that may result in markers not being moved correctly.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Variables to Conrefs

Creates conrefs from FM variables in the current file or book.

This command requires initial setup of the ConrefLibrary parameter in the *fm2dita.ini* file. Set the ConrefLibrary parameter to the name of the conref library file to be created or appended. This file is created in the same folder as the generated XML files.

When converting documents that contain variables, your conversion table must use the "UV:<varname>" syntax in order to map the variables to a specific element. In order to be processed by this command, the mapped element must assign the attribute "conrefid" with the value being that to be used as the ID of the conref. The conversion table entry for a "ProdName" variable would be as follows:

Wrap this object	In this element	With this qualifier
UV:ProdName	ph[conrefid="prodname"]	

After applying the conversion table, the variable would be wrapped in a <ph> element with the attribute @conrefid='prodname'. Once this command has been run, the @conrefid attribute will be replaced with the proper @conref attribute that references the element in the conref library file.

The conref library file is created if it doesn't already exist, then it is appended with a new entry for each variable definition (instance of a unique @conrefid in the source file). The conref source is created using a <p> tag container with a label that matches the @conrefid value followed by the inline element matching the element used by the variable. The ID attribute matches that of the @conrefid attribute. Each entry will follow this format:

```
<p>prodname: <ph id="prodname">DITA-FMx</ph></p>
```

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Build Menucascades

Creates a proper "menucascade" structure from a character-delimited sequence of elements.

This command requires initial setup in the BuildMenucascades section of the *fm2dita.ini* file. The following options must be set:

- SourceType - integer value that indicates the structure of content that you want to convert into a menucascade. If you are matching on a single element that contains multiple "menu items" separated with a delimiter, the SourceType is "0". If you are matching on multiple elements separated with a delimiter, the SourceType is "1".
- MatchSpec - element name or element/attribute specification used to locate each structure to process.
- Delim - specifies the character used to delimit each menu item.
- WrapperElem - specifies the element name to wrap the "menucascade" (typically menucascade).
- InnerElem - specifies the element name that is wrapped within the "menucascade" (typically uicontrol).

Refer to the information in Editing the fm2dita.ini File for details on these INI parameters.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26

Merge Code Lines

Combines sequential instances of the specified element into a single block in the current file or book.

This command prompts for an element name to merge. While processing, it locates multiple (2 or more) instances of the specified element and combines them into a single element of the same name. Each merged element is terminated with a line break (SHIFT+ENTER).

NOTE: *The attribute settings of the first element are migrated to the new container element, while any attributes on later elements will be lost.*

This is intended to be used to combine multiple separate instances of codeblock or similar elements into a single block element. If multiple element types need to be merged, run this command for each element type.

When checking for sequential elements, the outputclass attribute value is also compared. If the attribute values diffe, the elements won't be merged.

You can pre-set these values in the MergeElems parameter in the *fm2dita.ini* file. Refer to the information in Editing the fm2dita.ini File for details on this INI parameter.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49

Tab to Spaces

Replaces tabs with spaces in the specified element in the current file or book.

This command prompts for an element name in which to perform the tab to space replacement. If you need to replace tabs in multiple elements, run the command multiple times. Each tab is replace with the number of spaces indicated in the General/TabToSpaces parameter in the *fm2dita.ini* file. If no value is specified, 4 spaces are used.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49

Delete Invalid Attributes

Deletes the specified invalid attribute from the current file or book.

This command prompts for an attribute name to delete. If multiple attribute names need to be deleted, run this command for each attribute.

You can pre-set these values in the DeleteAttrs parameter in the *fm2dita.ini* file. Refer to the information in Editing the fm2dita.ini File for details on this INI parameter.

This command must be run after the EDD has been applied.

RELATED INFORMATION:
"FM2DITA Commands" on page 49

Delete Unstructured Markers

Deletes the specified unstructured marker from the current file or book.

This command prompts for an unstructured marker type to delete. If multiple marker types need to be deleted, run this command for each type.

You can pre-set these values in the DeleteMarkers parameter in the *fm2dita.ini* file. Refer to the information in Editing the fm2dita.ini File for details on this INI parameter.

This command must be run after the EDD has been applied.

RELATED INFORMATION:
"FM2DITA Commands" on page 49

Delete Empty Elements

Deletes empty elements of the specified type in the current file or book.

This command prompts for an element name to delete. While processing, it scans all elements of the specified name and checks for content. If the element contains nothing or only whitespace, it is deleted. If multiple element names need to be processed, enter multiple element names separated by spaces.

You can pre-set these values in the DeleteEmpty parameter in the *fm2dita.ini* file. Refer to the information in Editing the fm2dita.ini File for details on this INI parameter.

This command should be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Delete Elements" on page 67

Write Root Map (book)

Generates a DITA map from the current book which references submaps to chapters.

This command requires initial setup of the MapNameTpl parameter in the *fm2dita.ini* file as well as any other settings or properties that affect file naming (such as chapter numbering).

The map generated by this command is intended to be the root map when using book and chapter maps. By default, it is created as a standard DITA map; if you need a bookmap, retag the map accordingly or set the RootMapIsBookmap parameter to "1" in the *fm2dita.ini* file.

If a single map that references topics is needed, use the Write Single Map (book) command.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Topic and Map Template Building Blocks" on page 43
"Assign IDs to Topics" on page 66
"Write Root and Chapter Maps (book)" on page 81
"Write Single Map (book)" on page 81
"Write Chapter Map (file)" on page 82
"Write XML Topics" on page 83

Write Root and Chapter Maps (book)

Generates a DITA map from the current book which references submaps to chapters, also generates separate submaps for each chapter.

This command requires initial setup of the MapNameTpl and TopicNameTpl parameters in the *fm2dita.ini* file as well as any other settings or properties that affect file naming (such as chapter numbering).

This command generates both the root map as well as the submaps for each chapter. The submaps are created as a standard DITA maps. By default, the root map is created as a standard DITA map; if you need a bookmap, retag the map accordingly or set the RootMapIsBookmap parameter to "1" in the *fm2dita.ini* file.

If a single map that references topics is needed, use the Write Single Map (book) command.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Topic and Map Template Building Blocks" on page 43
"Assign IDs to Topics" on page 66
"Write Root Map (book)" on page 80
"Write Single Map (book)" on page 81
"Write Chapter Map (file)" on page 82
"Write XML Topics" on page 83

Write Single Map (book)

Generates a DITA map from the current book which references all topics in the chapter files.

This command requires initial setup of the MapNameTpl and TopicNameTpl parameters in the *fm2dita.ini* file as well as any other settings or properties that affect file naming (such as chapter numbering).

The map generated by this command is intended to be a single map for the entire book, directly referencing the topics in all files. It is created as a standard DITA map; if you need a bookmap, retag the map accordingly.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Topic and Map Template Building Blocks" on page 43
"Assign IDs to Topics" on page 66
"Write Root Map (book)" on page 80
"Write Root and Chapter Maps (book)" on page 81
"Write Chapter Map (file)" on page 82
"Write XML Topics" on page 83

Write Chapter Map (file)

Generates a DITA map for the current file.

This command requires initial setup of the TopicNameTpl parameter in the *fm2dita.ini* file as well as any other settings or properties that affect file naming (such as chapter numbering).

This command generates a map for the current file. The map is created as a standard DITA map; if you need a bookmap, retag the map accordingly.

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Topic and Map Template Building Blocks" on page 43
"Assign IDs to Topics" on page 66
"Write Root Map (book)" on page 80
"Write Root and Chapter Maps (book)" on page 81
"Write Single Map (book)" on page 81
"Write XML Topics" on page 83

Write XML Topics

Generates separate XML files for each topic in the current file or book.

This command requires initial setup of the TopicNameTpl parameter in the *fm2dita.ini* file as well as any other settings or properties that affect file naming (such as chapter numbering).

Each file is processed from the bottom, up. Each topic in the file is selected, cut and copied to a new empty file, then saved to XML. All files are written to the current file's folder using the structure application of the EDD that has been imported into the current file.

IMPORTANT: *Before running this command, all FM files should be validated using the* ***Element > Validate*** *command, for details see Task 4: Pre-export Validation.*

If you're using DITA-FMx, you should disable all "auto-prolog" options before running this command.

When processing is complete, any files left open indicate that something went wrong with the processing of that file. Note any content that remains in the file, but close it without saving (or save it to a new name).

This command must be run after the EDD has been applied.

RELATED INFORMATION:

"FM2DITA Commands" on page 49
"Editing the fm2dita.ini File" on page 26
"Topic and Map Template Building Blocks" on page 43
"Assign IDs to Topics" on page 66
"Write Root Map (book)" on page 80
"Write Root and Chapter Maps (book)" on page 81
"Write Single Map (book)" on page 81
"Write Chapter Map (file)" on page 82

3 Revision History

Describes the changes between versions of FM2DITA.

RELATED INFORMATION:

"1.01 - 29 November 2013" on page 92
"1.02 - 5 May 2014" on page 91
"1.03 - 30 April 2016" on page 87
"1.04 - 6 June 2017" on page 86

1.05 - 6 April 2020

New Features

Support for FM 2019

Sample File and Structure Application Updates

None

Bug Fixes / Minor Updates

None

1.04 - 6 June 2017

New Features

Support for FM 2017

Yes!

New or changed fm2dita.ini parameters

- General/MainFlow - new
- General/RootMapNameTpl - new
- General/StopWords - supports NULL value to prevent stripping of stopwords
- General/PunctuationChar - new

New building blocks and modifiers

- <$TITLE_SPACETODASH> - converts spaces to dashes in titles

Tables to Text command updates

Added support for complex mapping of table cells to paragraph tags. The UnwrapTablesPrefix INI parameter accepts a new syntax to allow finer control of tag mapping and assignment.

Sample File and Structure Application Updates

None

Bug Fixes / Minor Updates

Disallow duplicate dashes or underbars

When used for building filenames from titles, sequential dashes or underbars collapse into one character.

BookToDoc command fix

No longer includes the first file in the book twice.

Check For Topic Collisions command update

Now generates a document listing all proposed file names.

Generated map updates

Generated bookmap now includes frontmatter/toc elements.

RetagParas command update

The PREFIX parameter can now specify a replacement tag name rather than just a prefix. Use the "=" character to define the new tag name.

Move Markers command update

Supports adding of static attribute values.

RELATED INFORMATION:
"Revision History" on page 85

1.03 - 30 April 2016

New Features

New preconversion tools

- **Book to Doc** - Creates a single FM file from all files in a book.
- **Struct Cross-refs to Marker Cross-refs** - Relinks structure-based cross-references to marker-based cross-references.
- **Fix Cross-ref Formats** - Strips leading/trailing spaces from Cross-ref formats.
- **AFrame to Raster** - Generates raster images from anchored frames that contain multiple graphic objects.
- **List Import/Export Filters** - Generates a list of all import and export filters currently available in FrameMaker.

Updated preconversion tools

- **Retag Paras** - Allows the use of multiple retag para groups.

Removed preconversion tools

Replaced the Extract Art comand with AFrame to Raster.

fm2dita.ini file location changes

The *fm2dita.ini* file is now located by checking the current directory (the directory with the file or book being processed), then each parent directory. If it is not found the "default" *fm2dita.ini* file is used (the one in the

Pubs-Tools folder). This change allows you to maintain just one instance of an *fm2dita.ini* file for each project rather than needing to copy it to multiple folders.

New or changed fm2dita.ini parameters

- General/AssignIdElems - new
- General/TopicElems - changed default value to "" (nothing), now checks for @class contains "topic/topic".
- General/StopWords - added support for stopwords file
- General/FmDpiVal - new
- Added new AFrameToRaster section with the following parameters: ImageType, ImageFilenameTemplate, ImagePath, ImageDpi, and ConvertTypes.

Modified handling of lists in fm2dita.ini parameters

For "space-delimited" lists in *fm2dita.ini* parameters, if you need to use an alternate delimiter (other than the space character), use this new syntax:

[<char>]<char-delimited string>

Where <char> is the new delimiter enclosed in square brackets. For example, to use a comma as the delimiter, use the following:

[,]first item,second item,third item

New building blocks and modifiers

- <$IMG-DOCNAME> the document name (for AFrame to Raster command)
- <$IMG-IMGNAME> the first image file name in anchored frame [may be null] (for AFrame to Raster command)
- <$IMG-COUNT> the nth aframe processed (for AFrame to Raster command)
- <$VAR(*varname*)> value of the *varname* variable (for Map Hypertext Markers command or elsewhere)
- <$MARKERTEXT> value of the current marker's marker text (for Map Hypertext Markers command)
- <$LINKTEXT> value of the current marker's link text (for Map Hypertext Markers command)
- Added "split" building block modifier. [<index><splitchar>S] where <index> is 1-based, <splitchar> is a single character.

Retag Paras command updates

The Retag Paras command can now make use of multiple groups of rules for retagging paragraphs. This lets you define collections of rules that might be run for different situations or different types of content. Details are available in the Retag Paras topic.

Map Fix Images command updates

Added support for "fmdpi" in Fix Images command. If new INI parameter General/FmDpiVal is set, it assigns @outputclass='fmdpi:VAL', otherwise sets values for @height and @width attributes.

Map Hypertext Markers command updates

Added support for `gotolink filename:newlink` Hypertext markers. This assumes that all files are open at run time. It is possible to match on incorrect/duplicate file names, so use with care.

New API commands

- RELLINKS-TO-RELTABLE
- FLATTEN-CROSS-REFS
- BOOK-TO-DOC
- STRUCT-MARKER-CROSSREFS
- FIX-CROSSREF-FMTS

Updated API commands

- RETAG-PARAS - added optional <groupName> parameter.

Sample File and Structure Application Updates

Sample file changes

The conversion table and FM files have been updated to support conversion of related links to reltable entries.

Bug Fixes / Minor Updates

Fixed problems with stop words

Stop word processing should be working properly now.

If the result of processing leaves an empty string, now sets the string to the first "word."

Fixed "splitchar" operator for $URI building blocks

Now URI-encodes after the split is performed.

Report and log updates

Catalog Report command on FM12 and later, could go into infinite loop. This has been fixed.

Report and log files now display in a more user-friendly manner on FM12 and later.

Table to Text fixes

Fixed Table to Text command so it doesn't fail when multiple tables are anchored to the same paragraph. Also no longer deletes content in paragraphs that anchor tables that are deleted.

Properly converts table titles.

Fix Images fixes

Fixed Fix Images command so it moves <image> elements from <title> elements to the next sibling of the <title>.

If fig/@Id is empty, check the parent of fig for an @Id, if found, move it to fig/@Id.

No longer creates an empty @align attribute.

Book processing of images has been updated to "fix" all images.

Map Hypertext Markers

Works more consistently now.

Related Links to Reltables

Now scans all descendant elements of "elemname" (related-links).

Retag Paras fixes

Added support for "groups."

Merge Code Lines fixes

Now handles code lines that start with an inline element.

Delete Empty Elems fixes

Now deletes empty elements in tables.

Write XML Topics fixes

Updated support for shredding of glossentry topics, and any "non-title" topics.

RELATED INFORMATION:

"Revision History" on page 85

1.02 - 5 May 2014

New Features

Added the ability to move multiple marker types

The Move Markers command now supports moving multiple marker types. The MoveMarkers section in the INI file requires two new parameters, NumMarkerTypes and PrologElemPath. Additionally, if multiple markers are being moved, new MoveMarkers-*N* sections must be added for each marker type.

Added Map Hypertext Markers command

This command applies processing to "message URL", "gotolink", and "newlink" Hypertext markers. The "message URL" markers will convert into external xrefs, "gotolink" markers convert into external xrefs, based on the format specified in the GotolinkHref parameter, and "newlink" markers can optionally be converted into fm-data-marker elements.

Added Related Links to Reltable command

This command generates a reltable map from the related links in each file.

Allow file name reference of stop words

The StopWords INI parameter can now specify a plain text file for the list of stop words to remove from titles when generating topic file names.

New URI-encoding building blocks

Three new building blocks have been added: <$URI-VAR(*VARNAME*)>, <$URI-MARKERTEXT>, and <$URI-LINKTEXT>. These building blocks result in URI-coded values and are intended for use with the GotolinkHref parameter for the Map Hypertext Markers command.

Split the Fix/Flatten Cross-refs command into two commands

To support better processing of cross-refs and related links, the Fix/Flatten Cross-refs command has been split into two commands, Fix Cross-refs and Flatten Cross-refs.

Support disabling of date metadata in generated maps

By default, all generated maps will include the date in the topicmeta (or bookmeta). To disable this feature set the General/IncludeMetadataInMaps parameter to "0" in the *fm2dita.ini* file.

Sample File and Structure Application Updates

No changes or updates have been made to the structure application or sample files.

Bug Fixes / Minor Updates

Updated the Conversion Pod

The Conversion Pod has been updated to correspond with the changes to the available conversion commands.

RELATED INFORMATION:
"Revision History" on page 85

1.01 - 29 November 2013

New Features

Added modeless dialog "command pods"

On the new **Reports and Command Control** menu are the following new commands: **Preconversion Pod**, **Conversion Pod**, and **Write XML Pod**. These dialogs provide easy access to the associated commands.

Added commands for creating and editing the INI file

On the new **Reports and Command Control** menu are the following new commands: **Write INI for Document** and **Edit INI for Document**. These commands make it easier to work with the current project's *fm2dita.ini* file.

Added new Table to Text command

Converts tables of specified formats to text.

Added a new building block and building block modifiers

To support volumn numbering, you can use the $FM_VOLNUM building block. Now, in addition to numeric modifiers (for controlling the length of the text that results from a building block), you can include the modifiers 'U', 'L', and 'T', for "uppercase", "lowercase", and "title case".

Because of the new case modifiers, the case-specific building blocks (such as $FM_FILENAME_LC, and $TITLE_LC) have been removed from the

documentation, although they will still work as expected. Also removed the $T_MIN and $T_SEC building blocks from the documentation because of their limited usefulness.

Added new Delete Elements command

This command deletes the specified element(s).

Added the Map Hypertext Markers command

This is a partial implementation of supporting Hypertext markers. At this point, only the "message URL" syntax is supported.

Added the Build Menucascades command

Tool for wrapping uicontrol elements with a character delimiter (like ">") in a menucascade element.

Added programmatic control of FM2DITA commands

Most of the FM2DITA commands now have "API codes" which can be used with the FrameMaker FDK, ExtendScript, or FrameScript to automate a custom conversion process.

Sample File and Structure Application Updates

No changes or updates have been made to the structure application or sample files.

Bug Fixes / Minor Updates

Added the new Reports and Command Control menu

The report commands have been moved to this menu, and new commands have been added.

The Unwrap Elements command now unwraps table object elements

If a table object element (like tgroup) is specified for this command, that table will be unwrapped.

The XML writing commands create folders if defined by the filename template

When generating DITA topics or maps, if the filename template indicates a path for the file, that folder will be created if needed.

The Fix Tables command supports width and shading

Two new INI parameters, SaveTableWidth and SaveTableShading are available.

Chapter map filenames can now be based on the topic title

If your map template (MapNameTpl INI parameter) uses a building block based on the "title", it uses the root topic title in that file.

Fixed xref corruption on XML topic creation

In some cases, xrefs were getting corrupted. This seems to be fixed now.

Fixed problem that caused the last topic in a file from being written to XML

In some cases, the last (root) topic in a chapter file was not being written to XML. This has been fixed.

Update the Merge Code Lines command

This command now compares the value of the outputclass attribute to determine if lines should merge or not.

Root map can now be written as a bookmap

If the RootMapIsBookmap parameter is set to "1", the root map is written as a bookmap, if set to "0" it is written as a map.

Change default of MoveTblOutputclass INI parameter

The default value for MoveTblOutputclass is now "0" instead of "1".

RELATED INFORMATION:
"Revision History" on page 85

Index

U

V

X