

DocProcessor: Report Template Customization Guide

Revised: July 23rd, 2020

TopTeam, TopTeam Analyst, TopTeam Cloud, DocProcessor, Visual Process Designer and Visual Use Case are either registered trademarks or trademarks of TechnoSolutions Corporation in the United States and/or other countries.

Microsoft Word and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

This document is for informational purposes only. TECHNOSOLUTIONS MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Techno Solutions

Copyright © 2005-2020 TechnoSolutions Corp. All rights reserved.

Table of Contents

- Overview 5
 - What is it5
 - What Kind of Output it Produces5
 - How it works.....5
 - Who should read this document?6
 - Why would I need to customize a Report Template?.....6
- How DocProcessor works?..... 7
- Understanding Report Templates 8
- How to Fetch Records from Repository 9
 - Primary Fetch Commands.....9
 - Secondary Fetch Commands (for Linked Records)..... 10
 - Multi-Level Report using Fetch Commands 11
 - Auto-assigned Dataset Names 13
- Looping Through Fetched Records (Iterating Datasets) 14
 - EOF 14
 - BOF..... 15
- Outputting Field Values into Document..... 17
 - How it works..... 17
 - Field Tags..... 17
 - System Field Tags..... 17
 - Outputting Field Values using records in the fetched dataset 18
 - Generating content with Field Tags within Scan and EndScan 21
 - Syntax for Outputting Field Values of Different Types..... 23
 - Text Field Type*23
 - Number (Numeric) Field Type*23
 - Decimal Field Type*23
 - True / False Boolean Field Type*23
 - Date Field Type*.....23
 - Rich Text Field Type*.....23
 - Multi-Value Field Type*23
 - List Field Type*.....24
 - Date and Time (Timestamp) Field Type*24

Outputting Rich Text Field Value	24
Outputting Rich Text Field Value as Simple Text.....	25
Outputting Use Case Flows	25
Outputting Diagrams.....	26
<i>Master record</i>	26
<i>Child record</i>	26
Outputting System Field Tags.....	27
Outputting Sub-Report Field Values	28
<i>Comments</i>	28
<i>Attachments</i>	28
Outputting Charts	28
Applying Styles to Field Tags and Field Values	28
<i>Applying Styles to Commands</i>	29
Static Content.....	30
Understanding Report Query Context.....	30
Setting Context for the Report	30
<i>Set_Project</i>	30
<i>Set_Baseline</i>	32
<i>Set_Requirements_Document</i>	33
<i>Set_Release</i>	34
Understanding Variables	36
Declaring Different Types of Variables	36
SET(<Variable name>, <value>)	37
Using Report Parameters.....	38
Conditionally Including Report Sections	42
Conditional Constructs.....	45
IIF	46
Output only Non-Empty Fields.....	46
Check if field contains a Table	47
Grouping and Summaries	48
Miscellaneous Commands	50
Adding Comments within the template.....	50
Customizing Templates in TopTeam.....	51

What is needed to customize a Report Template?.....	52
Modifying Templates.....	53
How do I open Templates for customization?.....	54
Table of Contents	55
DocProcessor Limitations.....	59

Overview

What is it

DocProcessor™ is an easy-to-use, flexible and powerful engine for generating presentation quality documents and reports.

DocProcessor™ report generator is available in TechnoSolutions' applications: TopTeam Analyst, Visual Use Case, Visual Process Designer, etc.

What Kind of Output it Produces

DocProcessor generates Microsoft Word documents (*.docx), Acrobat Portable Document Format documents (*.pdf) or Rich Text Format (*.rtf) documents. Documents can contain rich text formatting, diagrams, and charts with almost any kind of layout and formatting.

You can produce formal documents such as System Requirements Specifications (SRS), Business Requirements Documents (BRD), Product Requirements Documents (PRD); you can also produce reports such as Requirements list, Use Case Detail report, etc.

DocProcessor is highly customizable and it can generate documents that comply with the most stringent templates or specifications such as your in-house IT standards, IEEE, ISO, FDA, CMMI, Engineering, Military, or Aerospace requirements documentation standards.

How it works

In order to use DocProcessor, you must first define a template. DocProcessor templates are ordinary Rich Text Format (*.rtf) files that can be easily customized by editing them in a word processor such as Microsoft Word®. Once the template is defined, you can then generate the desired document with a single click.

Who should read this document?

If you need to customize the reports and document templates that are shipped with the TechnoSolutions' products, or if you want to create new reports and document templates, you will find the information in this document helpful.

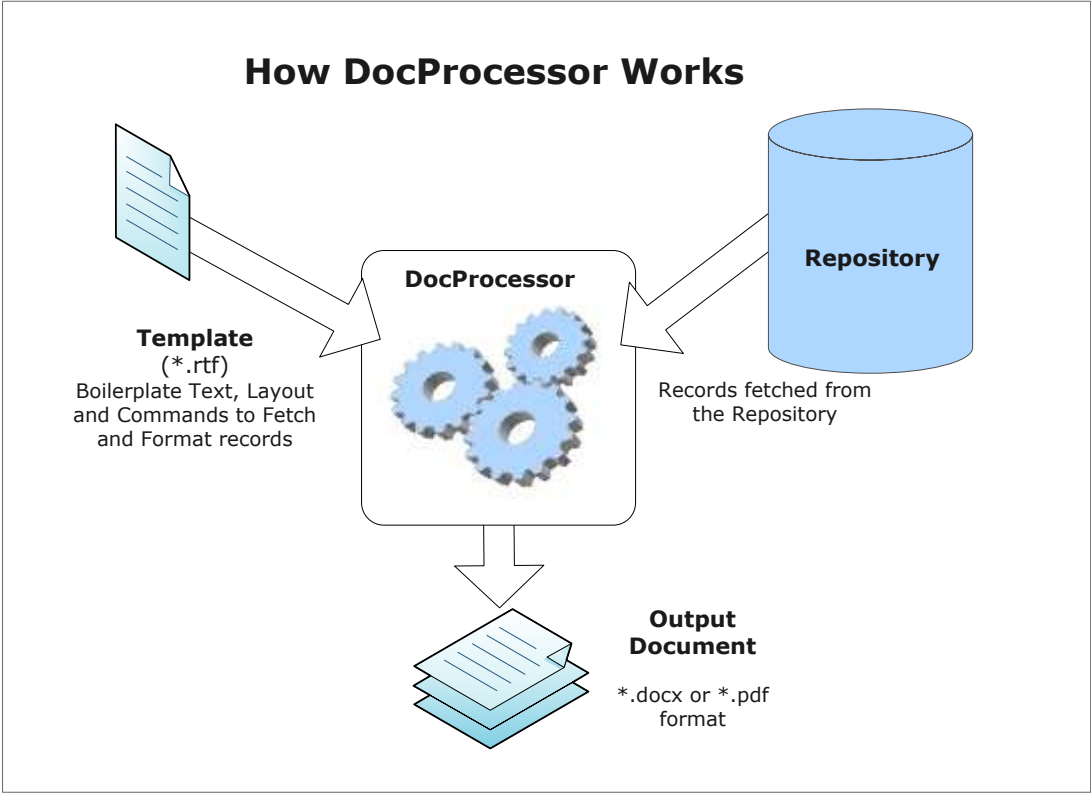
Why would I need to customize a Report Template?

You may need to customize a report or document template to:

- Create new reports in addition to the ones shipped with TechnoSolutions' products
- Add or remove fields and sub-reports from existing report templates
- Change the formatting of the output documents and reports as per your needs

How DocProcessor works?

DocProcessor reads a template file - which is a Rich Text Format file (*.rtf) - interprets and executes the commands present in the template to fetch data from the repository and outputs formatted results into the generated document.



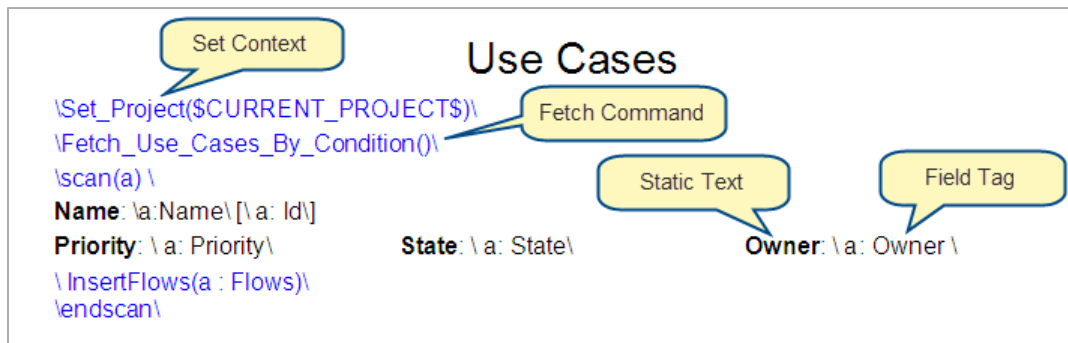
Understanding Report Templates

A report or document template is a Rich Text Format (RTF) file that can have:

1. Prompts for parameters that need to be supplied interactively at runtime
2. Commands to fetch data
3. Field tags for the repository fields to be generated (output) into a report
4. Static text
5. Static images

Template files are used by TopTeam's DocProcessor to generate reports. Template files have .rtf file extension and can be edited using any word processor such as Microsoft Word®.

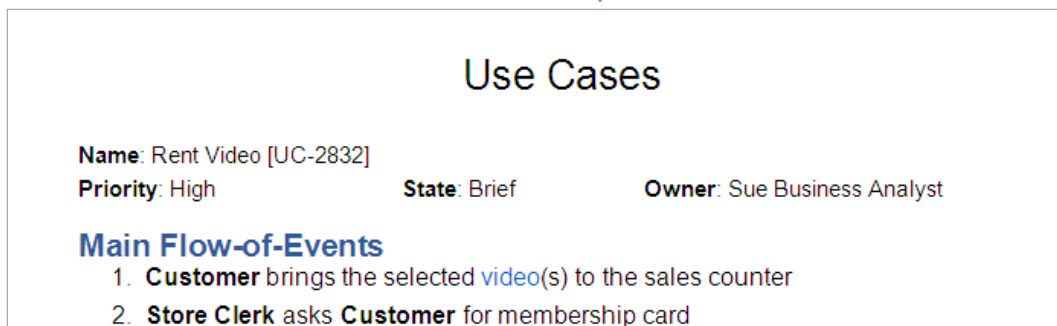
Basic Structure of a DocProcessor Template:



Sample RTF Report Template



Generate RTF Report



How to Fetch Records from Repository

One of the first things you need to do when creating or modifying a template is to decide what type of records you need from the repository and then place the corresponding [Fetch](#) command(s) in the template.

There are different types of [Fetch](#) commands available; some fetch records as a *Flat* list of records, while others fetch records in their original *Hierarchical (parent-child)* structure. Additionally, for a majority of *Fetch* commands, you can specify a filter and a sort order.

Once the records have been fetched from the repository, you can sequentially output field values from those records (via iteration) using [Scan](#) and [EndScan](#) constructs.

Primary Fetch Commands

Primary Fetch commands are used to fetch first level records in the repository such as *Requirements, User Stories, Use Cases, Screen Mockups*, etc.

Example Script:

```
\Set_Project($CURRENT_PROJECT$)\  
\Fetch_Use_Cases_By_Condition()\ \comment('<< This is a fetch command')\  
\scan(a) \
```

Name: \a:Name\ [\ a: Id\]

Priority: \ a: Priority\

State: \ a: State\

Owner: \ a: Owner \


```
\ InsertFlows(a : Flows)\
```


```
\endscan\
```

For more information on *Primary Fetch* commands, refer to the [DocProcessor Commands Reference Guide](#).

Secondary Fetch Commands (for Linked Records)

Secondary Fetch commands are used to fetch records that are in some way linked or associated with the primary records (primary records are fetched using *Primary Fetch* commands). These *Secondary Fetch* commands are placed inside a *Primary Fetch* command's **Scan/EndScan** loop.

IMPORTANT 	<i>Secondary Fetch</i> commands can only be placed inside a <i>Primary Fetch</i> command's iteration construct – i.e. Scan/EndScan
---	--

NOTE 	The Fetch and Scan commands below have been colored and made bold for clarity. This formatting is not required in normal usage.
--	---

Example Commands:

```
\Fetch_Comments_By_Condition("", 'Date')\  
\Fetch_Attachments_By_Condition("", 'File name')
```

Example Script:

```
\Set_Project($CURRENT_PROJECT$)\  
\Fetch_Use_Cases_By_Condition() \comment('<< This is the primary fetch command')\  
\scan(a) \
```

Name: \a:Name\ [\ a: Id\]

Priority: \ a: Priority\

State: \ a: State\

Owner: \ a: Owner \

```
\ InsertFlows(a : Flows)\
```

```
\Fetch_Comments_By_Condition("", 'Date') \comment('<< This is the secondary fetch')
```

Comments

Date	Person	Comment
------	--------	---------

```
\scan(b)\
```

```
\if (! eof(b))\
```

\ b : Date \	\b: Person \	\ b : Comment \
--------------	--------------	-----------------

```
\endif\
```

```
\endscan\ \comment('<< Endscan for the inner – secondary fetch')\
```

```
\endscan\ \comment('<< Endscan for the outer – primary fetch')\
```

For more information on *Secondary Fetch* commands, refer to the [DocProcessor Commands Reference Guide](#).

Multi-Level Report using Fetch Commands

Fetch commands can be placed inside an iteration construct (*Scan/EndScan*) of another *Fetch* command i.e. it is a nesting of *Fetch* and *Scan* commands. Such commands are typically used to fetch traceability records, recursively, resulting in multi-level traceability.

Example Script 1

```
\Set_Project($CURRENT_PROJECT$)\
```

```
\Fetch_Use_Cases_By_Condition()\
```

```
\scan(a) \
```

```
\a:Name\ [\ a: Id\]
```

```
\Fetch_Traced_Records_Of_Type_By_Condition('REQ', "", "", 'Trace Type, Type, Name')\
```

```
\scan(b)\
```

- \ b : Trace type \ - \ b : Name \ [\ b : Id \]

```
\Fetch_Traced_Records_Of_Type_By_Condition('TC', "", "", 'Trace Type, Type, Name')\
```

`\scan(c)\`

- `\ c : Title \ [\ c : Id \]`

`\endscan\`

`\endscan\`

`\endscan\`

Example Script 2

`\Fetch_Use_Cases_By_Condition ()\`

`\scan(a) \`

`\a:Name\ [\ a: Id\]`

`\Fetch_Traced_Records_of_Type_by_condition('FEAT')\if(!eof(b))\`

`\scan(b)\`

- `\ b : Title\ [\ b : Id\]`

`\Fetch_Traced_Records_of_Type_by_condition('FREQ')\if(!eof(c))\`

`\scan(c)\`

- `\ c : Title \ [\ c : Id \]`

`\Fetch_Traced_Records_of_Type_by_condition('TC')\if(!eof(d))\`

`\scan(d)\`

- `\ d : Title \ [\ d : Id \]`

`\endscan\endif\`

`\endscan\endif\`

`\endscan\endif\`

`\endscan\`

Auto-assigned Dataset Names

In the examples above, notice the `SCAN()` command. In the first `SCAN`, the parameter is `(a)`, in second it is `(b)`, in third `(c)`. Each nested `Fetch` command has a dataset name starting with the letter `"a"` and continues with `"b"`, `"c"`, `"d"`, etc. You can use the nesting level up to `"z"`.

NOTE



When using nested `Fetch` commands, you should consider the performance of a report, as well. The more you use nested `Fetch` commands, the more data will be fetched. Therefore, a larger report will be generated which may take longer to process.


Looping Through Fetched Records (Iterating Datasets)

Each [Fetch](#) command fetches records from the repository and places them in a dataset.

These records in the dataset are then iterated (looped) using the [SCAN](#) and [ENDSCAN](#) commands in order to access the field values in individual records. The records in the dataset appear in the sort order specified in the [Fetch](#) command.

The commands that you place within the [SCAN](#) and [ENDSCAN](#) tags such as field output tags, static text, secondary fetch, etc. **are executed for each record in the dataset.**

As you can see in the [Fetch](#) command examples, the datasets are named as "a", "b", "c", etc. (e.g. `\scan(a)`). In the template, you specify the dataset name in the [SCAN](#) command. (e.g. `\scan(c)`)

<p>IMPORTANT</p> 	<p>Dataset names need to be reused once the scope of that dataset is completed, i.e. all the records in the dataset have been iterated through to the end. That means the consecutive Fetch commands will have the same dataset name as the other Fetch commands at the same nesting level.</p>
---	--

EOF

End-of-File command is used to check whether the fetched dataset is empty (has no records) or whether you have reached the end of the dataset in the [Scan-EndScan](#) iteration process.

Syntax

`EOF(<Dataset identifier>)`

Example

1. `\if (eof(a))\` (Checking whether reached the end of the dataset a or whether it is empty)
2. `\if(!eof(a))\` (NOT end-of-file – checking that dataset has records and end of dataset not reached while iterating records)

Sample Code

```
\Set_Project('$CURRENT_PROJECT$')\
\Fetch_Use_Cases_by_Condition(' "State" = "All Open" ', 'Name')\ \comment('fetch command')\
```

Id	Name	State	Priority	Owner
-----------	-------------	--------------	-----------------	--------------

```
\scan(a)\ \comment('<< Start iterating fetched records / dataset ')\
\if (! eof(a))\ \comment('<< Check if all records processed. i.e. end of dataset ')\
```

```
\comment(' Below we are outputting field values from the current record ')\
```

\ a : Id\	\a:Name\	\ a : State\	\ a : Priority\	\ a : owner \
-----------	----------	--------------	-----------------	---------------

```
\endif\
\endscan\ \comment('<<< Go to the next record. scan-endscan always in a pair. ')\
```

BOF

Beginning-of-File command is used to check whether the dataset record pointer is pointing at the first record of the dataset.

Syntax

`BOF(DataSet)`

Example

1.\if(bof(a))\

2.\if(!bof(a))\

Sample Code

```
\Set_Project('$CURRENT_PROJECT$')\
```

```
\Fetch_Use_Cases_by_Condition(' "State" = "All Open" ', 'Name')\
```

Id	Name	State	Priority	Owner
-----------	-------------	--------------	-----------------	--------------

```
\scan(a)\
```

```
\if (bof(a))\
```

\ a: Id\	\a:Name\	\ a : State\	\ a : Priority\	\ a : owner \
----------	----------	--------------	-----------------	---------------

```
\endif\
```

```
\endscan\
```


Outputting Field Values into Document

Data that has been fetched from the repository into datasets can be output into the document.

How it works

Document contents are generated based on the *field tags* placed in a template.

DocProcessor parses (scans) the template, and when it finds a *field tag* corresponding to a field in the dataset it replaces the *field tag* with the value of that field in the current record.

The following types of field tags are available:

- **Field Tags**
- **System Field Tags**

Field Tags

Field Tags represent record fields fetched in the dataset. *Field Tags* are replaced with the actual field values when the report is generated.

For example:

Assuming you have already fetched Use Cases using the `\Fetch_Use_Cases()` command, you can output the names of the Use Cases in the dataset, by placing the tag "`\a:Name\`" in the template. When the report is generated, the tag `\a:Name\` will be replaced with the value of the field 'Name' for each record that is iterated (looped) within the `Scan-EndScan` for the Use Case dataset, which in this case is named `a`.

System Field Tags

System Field Tags are system defined tags that are used for inserting *non-dataset* or non-record values such as 'Today's Date', 'Current User', 'Project Name', etc.

For example: `\DATE\`, `\PROJECT_NAME\`

Example

```
\Set_Project($Current_Project)\  
\Fetch_Use_Cases_By_Condition()\  
\scan(a) \
```

```
\ a : Name \ \comment('<<< This will output value of field Name from dataset a ')\
```

```
\endscan\
```

```
Project: \PROJECT_NAME\ \comment('<<< This will output the name of the current project ')\
```

Outputting Field Values using records in the fetched dataset

Field Tags are used to output specific field values when a report is generated. A *Field Tag* represents a field of the current record. *Field Tags* are replaced with the actual field value when the report is generated.

All *Field Tags* must be entered in the format specified below:

```
\ <<dataset name>> : <<Name of the Field>> \
```

<<Dataset name>>

The prefix "a :" determines the dataset from which to pick up the field value.

Use prefix "a :" to output a field value for a Master record at the first nesting level.

Use prefix "b :" to output a field value for Secondary records at the second nesting level.


Use prefix "c :" to output a field value for third nesting level records, and so on.

<<Name of the Field>>

This is the caption of a field that you want to output. You can find out the caption of a field from the user interface of the corresponding editor in *TopTeam Analyst*, *Visual Use Case*, etc.

For example, to insert the tag for the field "Target Release" of a Use Case, the following tag is used:

\a : Target Release\

NOTE 	If the <<Name of the Field>> (field caption) in the template does not match the corresponding field caption in the repository, the generated output for the field will be empty.
--	--

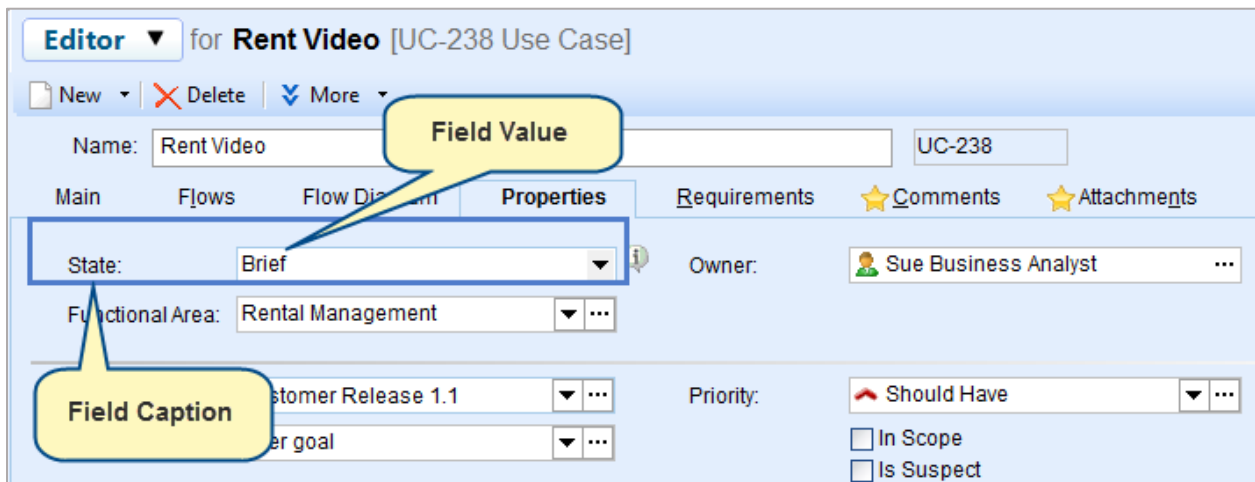
Example Field Tags

\a : Id \

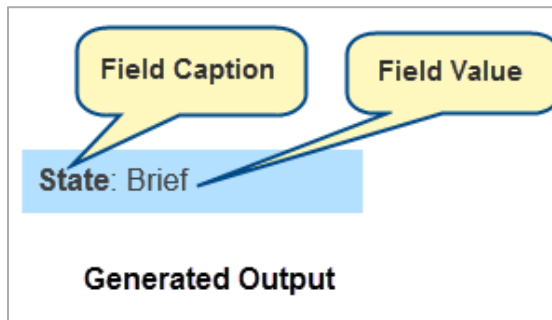
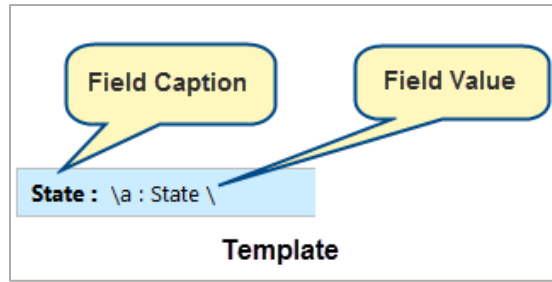
\a : Name \

\a : State \

The below images display *Field Caption* and *Field Value* in a record editor, template and generated output respectively:



The screenshot shows a record editor for a Use Case named "Rent Video" (UC-238). The interface includes a toolbar with "New", "Delete", and "More" options. Below the toolbar, there are tabs for "Main", "Flows", "Flow Diagram", "Properties", "Requirements", "Comments", and "Attachments". The "Properties" tab is active, showing fields for "State" (set to "Brief"), "Functional Area" (set to "Rental Management"), "Owner" (Sue Business Analyst), and "Priority" (Should Have). There are also checkboxes for "In Scope" and "Is Suspect". Two yellow callout boxes are present: one labeled "Field Value" pointing to the "State" dropdown menu, and another labeled "Field Caption" pointing to the "Customer Release 1.1" dropdown menu.



IMPORTANT



Only alphanumeric characters are allowed in *Field Tags*.

If the following non-alphanumeric characters are present in the *Field* caption, you **must replace these non-alphanumeric characters with spaces** when inserting the field tag.

, , ^, &, %, {, },), (, -, @, *, !, /, \, #, \$, >, ?, _ =, +, |, ", ;, ., ` , ~, ' , :

Incorrect Field Tags

Est. Effort
Effort Remaining (Hrs)

Correct Field Tags (with non-alphanumeric characters replaced by spaces)

Est Effort
Effort Remaining Hrs

NOTE

If you change field captions for a record type in the repository, you will need to make the corresponding changes in all existing templates. Otherwise, you will get an empty output value.

For more information about specific [Fetch](#) commands such as [Fetch_Use_Cases_by_Condition](#), etc. and the fields that are accessible with that command, refer to the [Fetch_Use_Cases_By_Condition](#) command in [DocProcessor Commands Reference Guide](#).

Generating content with Field Tags within Scan and EndScan

Field Tags that are placed within the `\Scan()` and `\EndScan` command, will be repeatedly generated for all records present in the dataset.

```
\Scan(DataSet) [,page] [,!eof]\
```

<< *Place your field tags here* >>

```
\EndScan\
```

The option "page" forces to begin every record of the scanned dataset (besides the first) from a new page.

Example

```
\scan(a) [,page]\
```

Example

```
\Set_Project('$Current_Project$')\  
\Fetch_Repository_Objects_By_Condition('MSG', "", "")\  
\scan(a) [,page]\
```

\a : Title\

```
\Insert_Permalink(a: ID)\  
\Fetch_Attachments_By_Condition("", 'File name')\  
\
```

Attachments

File Name	Person	Added on
\scan(b) \		
\ b : File name \	\b : Person \	\b:Added on \
\endscan\		
\endscan\		

When the option “!eof” is used, the report generator will skip entire scan block if the scanned dataset has no records. This option is useful while generating master-detail reports.

Example

```
\scan(a) [,!eof]\
```

Example

```
\Set_Project('$Current_Project$')\
\Fetch_Repository_Objects_By_Condition('MSG', '', '')\
\scan(a) [,!eof]\
```

\a : Title\

```
\Insert_Permalink(a: ID)\
\Fetch_Attachments_By_Condition("", 'File name')\
```

Attachments

File Name	Person	Added on
\scan(b) \		
\ b : File name \	\b : Person \	\b:Added on \
\endscan\		
\endscan\		

Syntax for Outputting Field Values of Different Types

Text Field Type

`\ a : Name \`

Number (Numeric) Field Type

`\ a : Days Open \`

Decimal Field Type

`\ a : Est Effort Hrs \`

True / False Boolean Field Type

`\ a : Locked \`

Date Field Type

`\ a : Crt dt \`

Rich Text Field Type

`\ InsertRtf(a : Description) \`

`\ Insert_Rich_Text_Using_Word (a : Steps) \`

See the next section for detailed information on how to use the rich text commands.

Multi-Value Field Type

`\ a : Multi value Field \`

List Field Type

```
\ a : Priority \
```

Date and Time (Timestamp) Field Type

```
\ a : Date and Time \
```

Outputting Rich Text Field Value

Rich text fields (including HTML fields) need special syntax to output value into the document.

Use the following tag syntax to output Rich Text fields:

```
\ InsertRTF(<<Field Tag for a Rich Text Field>>)\
```

```
\ InsertRtf(a : RichTextFieldCaption) \
```

For example, to insert the description of a record, use the following tag:

```
\ InsertRtf(a : Description) \
```

IMPORTANT



`InsertRTF()` command does not support **outputting nested tables** i.e. a table inside a cell of another table.

For Rich Text Field that has **nested tables or strike-out text style**, use the `Insert_Rich_Text_Using_Word()` command.

For more information, refer to the `Insert_Rich_Text_Using_Word` command in the section *Commonly used commands* in [DocProcessor Commands Reference Guide](#).

Example

```
\ Fetch_Actors_By_Condition ()\
```

```
\ scan(a) \
```



```
\ a:Name \ [a: Id\]
```

```
\ InsertRtf(a : Description) \
```

```
\ endscan \
```

Outputting Rich Text Field Value as Simple Text

This command is only applicable to Rich Text Fields.

```
\ InsertRtfAsText (<<Field Tag for a Rich Text Field>>)\
```

This command converts Rich Text Field data into plain text and outputs into the document being generated.

```
\ InsertRtfAsText (a : Description)\
```

Example

```
\Fetch_Use_Cases_By_Condition()\
```

```
\scan(a) \
```

```
\ a:Name \ [a: Id\]
```

```
\ InsertRtfAsText(a : Pre conditions)\
```

```
\endscan\
```

Outputting Use Case Flows

This command outputs the Use Case Flow-of-events of a Use Case.

```
\ InsertFlows(<<Field Tag for Use Case Flows Field>>)\
```

For example, to insert the Main Flow-of-Events of a Use Case, use the following tag:

```
\InsertFlows(a : Flows)\
```

Example

```
\Fetch_Use_Cases_By_Condition ()\
```

```
\scan(a) \
```

```
\ a:Name \ [a: Id\]
```

```
\ InsertFlows(a : Flows)\
```

```
\endscan\
```

Outputting Diagrams

Use this command to output diagrams such as Use Case Diagrams, Context Diagrams, Screen Mockups, Business Processes, etc.

This command is applicable only to diagram fields.

```
\Insert_Diagram_Custom(<<Field Tag for Diagram Field>>,'<<Width>>','<<Height>>',  
'<<Image Size Unit>>','<<Image_Type>>')
```

Master record

```
\ Insert_Diagram_Custom ( a : Diagram ) \
```

Child record

```
\ Insert_Diagram_Custom ( b : Diagram ) \
```

For more information, refer to the section *Insert Diagram* in [DocProcessor Commands Reference Guide](#).

Outputting System Field Tags

System Field Tags are system-defined tags that are used for inserting values such as 'Today's Date', 'Current User', etc.

FILE_NAME	Outputs the name of the file that is generated.
PROJECT_NAME	Outputs the project name set using the Set_Project command.
PROJECT_PATH	Outputs the entire project folder path of the project set using the Set_Project command.
CURRENT_DATE	Outputs the system date on your computer.
USER_NAME	Outputs the login name of the current <i>TopTeam</i> user.
COMPANY_NAME	Not in use (Reserved for future).
REQUIREMENTS_DOCUMENT_NAME	Outputs the Requirements Document Name that is set using the Set_Requirements_Document command.
RELEASE_NAME	Outputs the Release Name that is set using the Set_Release command.
BASELINE_NAME	Outputs the Baseline Name that is set using the Set_Baseline command.
PRODUCT_NAME	Outputs Product Name (example TopTeam / Visual Use Case).
PRODUCT_VERSION	Outputs Product Version (Desktop App Version).
PRODUCT_EDITION	Outputs Product Edition Type.
CRT_DT	Outputs date on which a record was created.
UPD_DT	Outputs date on which a record was last updated.
CRT_BY	Outputs the login name of a user who created the record.
UPD_BY	Outputs the login name of a user who last modified the record.
VERSION_NUMBER	Outputs the version number of a record.

Outputting Sub-Report Field Values

Alphabet "b :" is used to identify *Sub-Report* fields for the parent first level records. A typical construct for a *Sub-Report* is: "Comments", "Attachments" sub-reports.

Comments

```
\ b: Person \ \ b: Comments \
```

Attachments

```
\ b: Person \ \ b: File Name \
```

Outputting Charts

This command is used to generate Dashboard Chart in an output template. The first parameter is the name of the chart portlet that you define in the Dashboard.

```
Insert_Chart_Using_Portlet('<<Chart Portlet Name>>', '<<Width>>', '<<Height>>', '<<Image Size Unit>>', '<<Image Type>>')
```

Example

```
\ Insert_Chart_Using_Portlet ('Requirements Assigned to Me ( by State )', 6, 7, 'INCHES', 'JPEG') \
```

Applying Styles to Field Tags and Field Values

You can format the output of fields by applying styles to *Field Tags*. The output value automatically takes the style that has been applied to the *Field Tag*. You can also apply styles such as Heading1, Heading2, Heading3, etc. which can be helpful in generating the Table of Contents.

Example

```
\Fetch_Requirements_By_Condition("", 'Title')\
\scan(a) \
```

Id(without Style): \a: Id\

Id(with Style): \ **a: Id** \

Title(without Style): \a:Title\

Title(with Style): \ **a:Title** \

Source (without Style): \ a : Source \

Source (with Style): \ **a : Source** \

```
\endscan\
```

NOTE



Since the data in Rich Text Fields is already formatted when entering the text, you cannot “re-style” the contents of a Rich Text Field when it is generated into a report i.e. a Rich Text Field is generated with the same formatting and style that you originally entered when authoring the field.

Applying Styles to Commands

We recommend that you apply “Command” style to all the commands used in the report. If the “IF” command is not applied with “Command” style, it may add an empty line in the report.

Example 1

```
\Fetch_Repository_Objects_By_Condition('DD', "", 'Name')\
```

Title	ID	Active State(s)
\ a : Name \	\Insert_Permalink(a : ID)\	\Insert_Multi_Value_Field(a : Active States,', ')\

```
\endif\
\endscan\
```

Example 2

```
\Fetch_Repository_Objects_By_Condition('DD', "", 'Name')\
```

Title	ID	Active State(s)
\a : Name \	\Insert_Permalink(a: ID)\	\Insert_Multi_Value_Field(a : Active States, ')\

NOTE: In example 2, style is applied to the `\if(!eof(a))\` command.

Static Content

Static Content is used for headings, captions, separator lines, images, tables, etc. *Static Content* in the template is generated exactly as entered by the author without further processing or alteration.

Understanding Report Query Context

Setting Context for the Report

The `Set` commands set the “**context**” under which the subsequent `Fetch` commands operate. For example, when you use the `Set_Project()` command, all subsequent `Fetch` commands will fetch records from the selected Project. Another example: If you use the `Set_Baseline()` command, all subsequent `Fetch` commands will fetch only those records that were included in the selected Baseline.

Set_Project

A *Set Project* sets the context of the document to the specified Project.

Syntax

```
\Set_Project('<<Project Path>>')\
```

NOTE

If you want to fetch records for more than one Project in the same document, you can specify multiple `Set_Project()` commands sequentially. The `Fetch` commands that appear after a `Set_Project` command will fetch data from the corresponding Project.

Example

```
\Set_Project('Video Rental System')\
```

```
\Set_Project('Video Rental System\Reports')\
```

```
\Set_Project('$CURRENT_PROJECT$')\
```

This sets the context of the document to the current Project in the application. This is the most commonly used command.

```
\Set_Project('$PARENT_OF_CURRENT_PROJECT$')\
```

This sets the context of the document to the parent Project of the current Project in the application. This command is useful when you want to fetch records from the parent Project of the current Project.

```
\Set_Project('$ROOT_OF_CURRENT_PROJECT$')\
```

This sets the context of the document to the root Project, or top-level Project, of the current Project in the application.

```
\Set_Project('$PROMPT_PROJECT$')\
```

This command prompts you to interactively select the Project when the document is generated. This command displays a Project selection screen at run-time when it encounters this command.

```
\Set_Project_By_Id('<<Project ID>>')\
```

This sets the context of the document to the specified Project ID.

Example

```
\Set_Project('$CURRENT_PROJECT$')\
```

```
\Fetch_Use_Cases_By_Condition ()\
```

```
\scan(a) \
```

```
\ a:Name \ [a: Id\]
```

```
\ InsertFlows(a : Flows)\
```

```
\endscan\
```

Set_Baseline

This sets the context of the document to the specified baseline. Subsequent [Fetch](#) commands will fetch only those records that are included in the specified Baseline.

Syntax

```
\Set_Baseline('< <Baseline Name>>')\
```

IMPORTANT



The Baseline specified as a parameter to this command **MUST** exist in the Project that was set previously in the template using the [Set_Project\(\)](#) command.

Example

```
\Set_Baseline('Test Release 1')\
```

```
\Set_Baseline ('$PROMPT_BASELINE$')\
```

This command prompts you to interactively select the Baseline when the document is generated. This command displays a Baseline selection screen at run-time when it encounters this command.

Example

```
\Set_Project('$CURRENT_PROJECT$')\  
\Set_Baseline ('$PROMPT_BASELINE$')\  
\Fetch_Use_Cases_By_Condition ()\  
\scan(a) \  
  
\ a:Name \ [a: Id\  
  
\ InsertFlows(a : Flows)\  
\endscan\  

```

Set_Requirements_Document

This sets the context of the document to the specified Requirements Document. Depending upon the Requirements Document set by you, subsequent Requirement's [Fetch](#) commands will fetch data for that Requirements Document.

Syntax

```
\Set_Requirements_Document('<<Requirement Document Name>>')\  

```

Example

```
\Set_Requirements_Document('$DEFAULT_REQUIREMENTS_DOCUMENT$')\  

```

If the above command is used with the default parameter, you need to first use the [\Set_Project\('<Project>'\)](#) command in order to set the Project in the current context. To fetch the records of the requirements document, use [\Fetch_Requirements_Tree_By_Condition\(\)](#) command.

```
\Set_Requirements_Document('Bussiness\Req Doc 1')\  
\Set_Requirements_Document('Req Doc 1')\  
\Set_Requirements_Document()\  

```

```
\Set_Requirements_Document('$PROMPT_REQUIREMENTS_DOCUMENT$')\
```

If this command is used with “\$PROMPT_REQUIREMENTS_DOCUMENT\$” and has no parameters, a pop-up window will display for the Requirements Document selection. In this case, you don’t need to set the Project. If you have set the Project, then a pop-up will show the Requirements Document selection option, only to you.

Example

```
\Set_Project('$CURRENT_PROJECT$')\  
\Set_Requirements_Document('$PROMPT_REQUIREMENTS_DOCUMENT$')\  
\PROJECT_NAME\  
  
\Fetch_Requirements_Tree_By_Condition( )\  
\scan(a) \  
  
\a: wbs \ \ a : Title \ [ \ a : Id \ ]  
  
\endscan\  

```

Set_Release

This sets the context of the document to the specified Release. Subsequent [Fetch](#) commands will fetch records that are included in the specified Release.

Syntax

```
\Set_Release('<<Release Name>>')\  

```

IMPORTANT



The release specified as the parameter to this command **MUST** exist in the Project that was set previously in the template. If the release specified here is not defined in that Project, the release command will be ignored.

Example

```
\Set_Release('Alpha')\  
\Set_Release('')\  
\Set_Release('$PROMPT_RELEASE$')
```

If the argument passed is '\$PROMPT_RELEASE\$' or no parameter is passed, then a pop-up screen for the release selection appears. If Project is not set, the Project selection option is also available in that window. If Project is already set before executing this command, only Release may be selected.

Example

```
\Set_Project('$CURRENT_PROJECT$')\  
  
\Set_Release('$PROMPT_RELEASE$')\  
\Fetch_Use_Cases("", 'Name')\  
\scan(a) \  
  
\a:Name\ [\ a: Id\  
  
\endscan\  

```

For more information, refer to the section *Set and Clear Context Commands* in [DocProcessor Commands Reference Guide](#).

Understanding Variables

Variables are declared and used in DocProcessor Templates to dynamically customize the template using report parameters. Variables can be used to perform mathematical calculations, can be used in IF-ELSIF-ELSE-ENDIF constructs, can be used to create dynamic filter conditions, etc.

Declaring Different Types of Variables

You can declare different types of variables using the following commands:

```
\VAR()\n\Declare_Variable()\n\Declare_Variable_State()\n\Declare_Variable_LOV()\n\Declare_Variable_ID()\n\Declare_Variable_Name()\n\Declare_Variable_Baseline()\n\Declare_Variable_Release()\n\Declare_Variable_Project()\n\Declare_Variable_Folder()\n\Declare_Variable_User()\n\Declare_Variable_Record_Type()\n\Declare_Variable_Record_Type()\n\Declare_Variable_Filter()\n\Declare_Variable_Custom_Value()\n\Declare_Variable_Package_Baseline()\n\Create_Sections('<<Section1>>', '<<Section2>>', '<<Section3>>', '<<Section4>>',  
'<<Section5>>',....., '<<Section N>>')\
```

For more information on these commands, refer to the section *Declare Variable* in [DocProcessor Commands Reference Guide](#).

SET(<Variable name>, <value>)

Example

```
1.\SET>LastLevel, a : Level)\  
2.\SET>Lastestcost, a:Est Cost)\
```

Sample Code

```
\Set_Project('$CURRENT_PROJECT$')\  
\Fetch_Use_Cases_By_Condition('', 'Level,Name')\  
\VAR>LastLevel) \\ LastLevel:='\  
\scan(a)\  
\if (LastLevel <> a : Level)\  
  
Level : \a: Level\  
  
\endif\  
  
\a:Name\ [\a:Id\  
  
\SET>LastLevel, a : Level)\  
\endscan\  

```

Using Report Parameters

The *Prompt* command is used to customize the output of a template by allowing you to choose options and parameters at run time.

You can define variables and prompt for their values at run time. Based on what you enter or choose for the corresponding parameter values, report sections can be turned ON or OFF using IF constructs. This way you can have the same template produce a variety of different reports without the need to create multiple templates.

Syntax

```
\Prompt_For_Variable_Values()  
\Prompt_For_Variable_Values(<< Variable1>>, <<Variable2>>,<<Variable3>>,....  
<<VariableN>>)\
```

This prompts the dialog for SELECTED variables.

Syntax

```
\Prompt_For_Variable_Values('$ALL$')\
```

Parameter **\$ALL\$**: This prompts for setting the values of **ALL** variables created using **Declare Variable()**.

Variables are created using **Declare Variable()** commands.

Sections are created using **Create_Sections()** command.

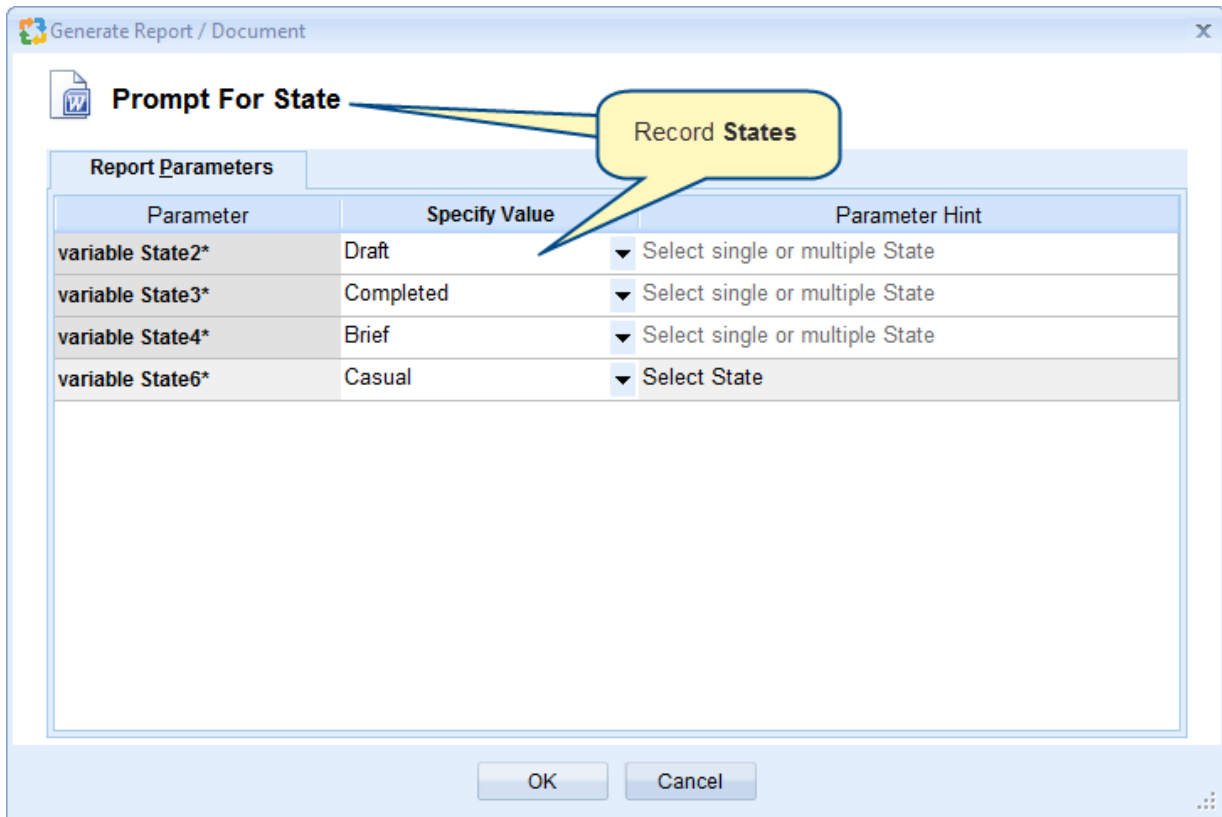
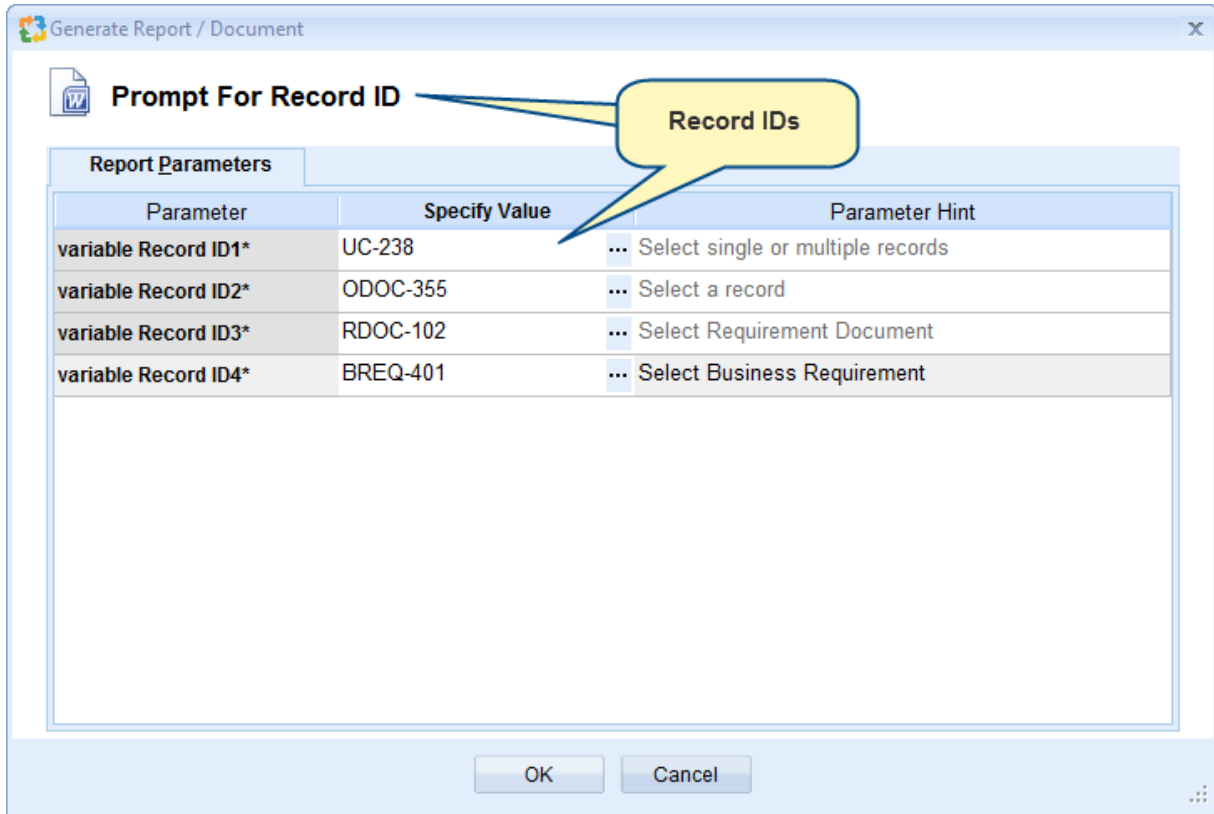
The **Prompt_For_Variable_Values** command is used to set the value of Variables or Sections while the report is generating.

NOTE

`Prompt_For_Variable_Values` command should always be preceded with `Declare Variable` or `Create_Sections`.

The Prompt window appears as shown below. Based on the number of variables or sections created, this window displays all those variables.

Parameter	Specify Value	Parameter Hint
Record ID*	RDOC-162	Enter Requirement Document Display Id
Filter Name*		Enter Filter Name to filter Requirement Document
WBS Code*	1	Enter Wbs Code for Requirement Document records
Show Parent*	<input checked="" type="checkbox"/>	Check Show Parent option to displayHeader records



Example 1

```
\Set_Project('$CURRENT_PROJECT$')\  
\Declare_Variable('Variable_ID', 'Number', 'Enter Requirements Document ID')\  
\Declare_Variable('Variable_Filter_Name', 'Text', 'Enter Filter Name created in Requirements tree')\  
\Prompt_For_Variable_Values('Variable_ID', 'Variable_Filter_Name')\  
\Fetch_Requirements_Tree_By_Document_Id(Variable_ID, Variable_Filter_Name)\  
\scan(a) \
```

```
\a: wbs \ \ a : Title \ [ \ a : Id \ ]
```

```
\endscan\  

```

Example 2

```
\Declare_Variable('Variable_ID', 'Number', 'Enter Requirements Document display Id')\  
\Declare_Variable('Variable_Filter_Name', 'Text', 'Enter Filter Name created in Requirements tree')\  
\Prompt_For_Variable_Values('$ALL$')\  
\Fetch_Requirements_Tree_By_Document_Id(Variable_ID, Variable_Filter_Name)\  
\scan(a) \
```

```
\a: wbs \ \ a : Id \ \ a : Title \
```

```
\endscan\  

```

Conditionally Including Report Sections

`\ Create_Sections()\`

This command is used to create multiple Boolean variables representing report sections in a single `Create_Sections` command.

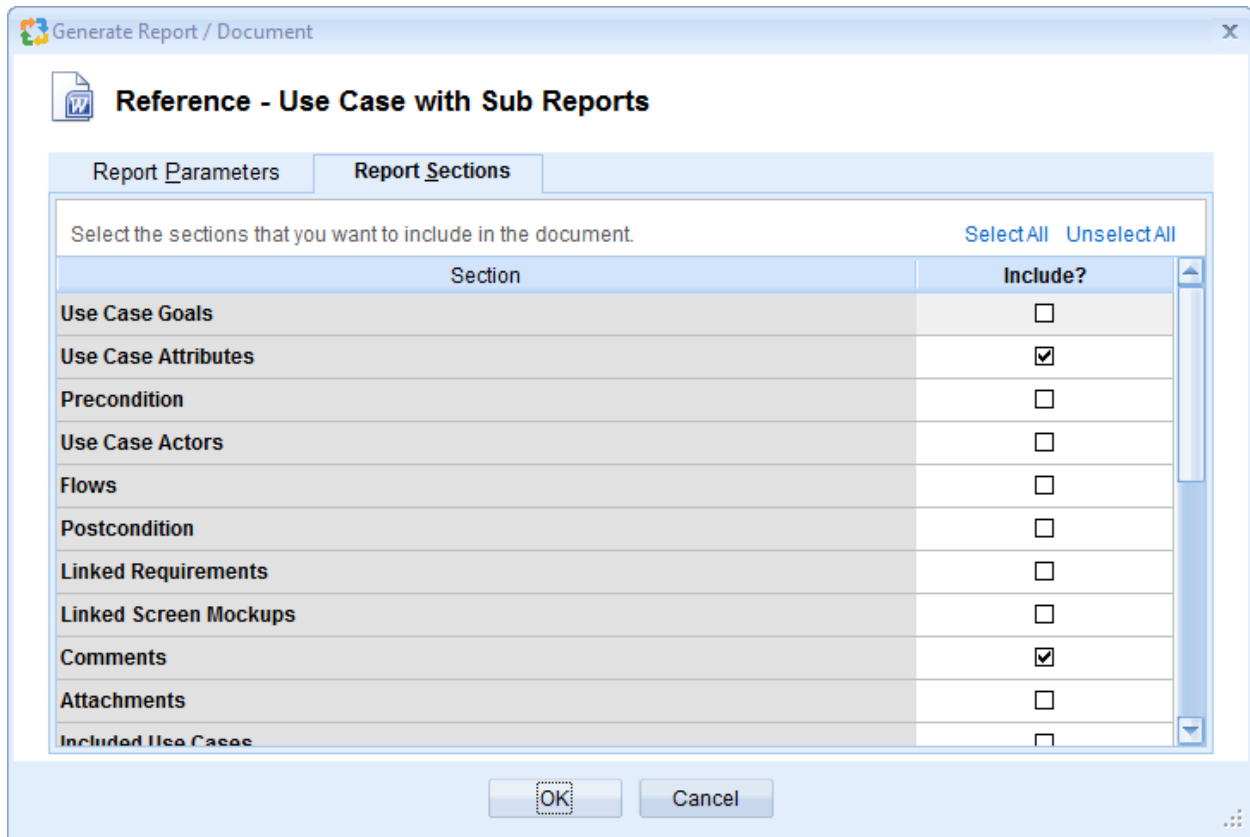
The `Prompt_For_Variable_Values` command allows you to include or exclude sections when the report is generated.


Syntax

```
\Create_Sections('<<Section1>>', '<<Section2>>', '<<Section3>>', '<<Section4>>',  
'<<Section5>>',....., '<<Section N>>')\
```

<<Section>>: Specify name of the section that you want to define.

By default, all the sections in the prompt will appear unselected. You can select the required sections. Depending on how the Boolean variables are used in the template in IF-ELSIF-ELSE-ENDIF constructs, the template output will be generated.



<p>NOTE</p> 	<p>The Create_Sections command is always used in conjunction with Prompt_For_Variable_Values.</p>
--	---

Example

```

\Set_Project('$CURRENT_PROJECT$)\
\Create_Sections( 'Section_Use_Cases', 'Section_Screen_Prototype', 'Section_Glossary')\
\Prompt_For_Variable_Values('$ALL$)\

\if(Section_Use_Cases) \
\Fetch_Use_Cases(" , 'Name')\

```

Use Cases

```
\scan(a) \
```

```
\a:Name\ [\ a:Id\]
```

```
\endscan\
```

```
\endif\
```

```
\if(Section_Screen_Prototype) \
```

```
\Fetch_Diagrams ('SCR',' ', 'Name')\
```

Screen Prototypes

```
\scan(a) \
```

```
\a:Name\ [\ a:Id\]
```

```
\endscan\
```

```
\endif\
```

```
\if(Section_Glossary) \
```

```
\Fetch_Glossary()\
```

Glossary

```
\scan(a)\
```

```
\a:Term\
```


```
\endscan\
```

```
\endif\
```

Conditional Constructs

If you want certain commands to be executed, certain fields, or a certain report section, to be generated only when a particular condition is true, use the following conditional constructs:

'IF', 'ELSEIF' and 'ENDIF'.

<p>NOTE</p> 	<p>DocProcessor does not support the nested "IF...ELSE" control statements within "IF...ELSE".</p> <p>Also, while using the "IF" statement, it executes case-sensitive string comparison.</p>
--	---

`\if(<boolean expression>)\`

<<Text and commands to be processed in case above is true>>

`\elseif(<boolean expression >)\`

<<Text and commands to be processed in case above is true>>

`\else\`

<<Text and commands to be processed in case all above conditions fail >>

`\endif\`

`\elseif` and `\else\` are optional.

`<boolean value>` may be a variable, data field or DocProcessor command that returns a Boolean value.

IIF

Syntax

```
\ IIF(Logical_expr, Value1, Value2) \
```

Example

```
\ IIF((a>b),a,b) \
```

Sample Code

```
\var(a)\ \a := 20\
```

```
\var(b)\ \b := 25\
```

Greatest among the two numbers

```
a = \a\
```

```
b = \b\
```

```
Greater = \IIF((a>b),a,b)\
```

Output only Non-Empty Fields

```
\ If (Is_Field_Non_Empty (<<Name of the Field>>)) \
```

This command is TRUE if the field contains data or is non-empty. Use this command in conjunction with [IF\(\)](#) statement to decide which fields to output and which ones to skip.

Example

```
\Fetch_Requirements_By_Condition("",'Title')\
```

```
\scan(a) \
```

```
\a:Title\ [\a: Id\]
```

```
\if (Is_Field_Empty (a : Description))\
```

Description not available

```
\endif\
```

```
\if (Is_Field_Non_Empty (a : Description))\
```

```
\ InsertRtf (a : Description) \
```

```
\endif\
```

```
\endscan\
```

Check if field contains a Table

```
\if(Does_Text_Contain_Table(a:Description))\
```

This command is used to check if a table is present in the specified Rich Text Field.

Example

```
\Set_Project('$CURRENT_PROJECT$')\
```

```
\Fetch_Requirements_Tree_By_Document_Id_By_Condition('$DEFAULT_REQUIREMENTS_DOCUMENT$')\
```

```
\scan(a)\
```

```
\if(Does_Text_Contain_Table(a:Description))\
```

```
\a: wbs \ \a : Title \ [ \ a : Id \ ]
```

```
\InsertRTF(a: Description)\
```

```
\endif\
```

```
\if( ! Does_Text_Contain_Table(a:Description) ) \
```

<pre>\a: wbs \</pre>	<pre>\ a : Title \ [\ a : Id \]</pre>
<pre>\ InsertRtf(a : Description)\</pre>	

```
\endif\  
\endscan\  

```

Grouping and Summaries

Records can be displayed and grouped by a specific field using a report variable. The report variable is used to hold the value of the field on which grouping is needed. Here, the report variable "LastPriority" is used to hold a value of the Priority field.

The following example shows records group by the Priority field:

Example

```
\Set_Project('$CURRENT_PROJECT$')\  
\Fetch_Use_Cases_By_Condition('', 'Priority, Name')\  
  
\VAR(SubEfforts, SubCount, TotalCount, TotalEffort) \  
\ TotalEffort:=0\  
\ SubEfforts:=0\  
\ SubCount:=0\  
\ TotalCount:=0\  
\VAR(LastPriority) \  
\ LastPriority:=""\  

```

Name	State	Owner	Est Effort (Hrs)
------	-------	-------	------------------

```
\scan(a) \  
\ IF(LastPriority <> a : Priority) \if (SubEfforts <> 0) \  

```

Sub Count = \ SubCount \	Sub Total: \ SubEfforts \
---------------------------------	----------------------------------

```
\ SubCount:=0 \ SubEfforts:=0 \  
\endif \  

```


Priority: \ a : Priority \	Est Effort (Hrs)
-----------------------------------	-------------------------

\endif\

\ a : Name \ [a : ID]	\ a : State \	\ a : owner \	\ a : Est Effort Hrs \
-------------------------	---------------	---------------	------------------------

\ TotalCount:= TotalCount +1\ SubCount:= SubCount +1\ SubEfforts:= SubEfforts + a : Est Effort Hrs \ TotalEffort:= TotalEffort + a : Est Effort Hrs \ SET(LastPriority, a : Priority)\ endscan, sum(a: Est Effort Hrs, TotalEffort)\

Sub Count = \ SubCount \	Sub Total: \ SubEfforts \
Count= \ TotalCount \	Total Est Effort:\ TotalEffort \

Report output will look such as below:

Name	State	Owner	Est Effort (Hrs)
Priority: High			Est Effort (Hrs)
Rent Video [UC-2832]	Brief	Sue Business Analyst	4
Return Video [UC-2834]	Completed		36
Sub Count = 2		Sub Total: 40	
Priority: Normal			Est Effort (Hrs)
Calc Frequent Renter Discount [UC-2838]	Implemented	Chris Team Lead	48

Miscellaneous Commands

Adding Comments within the template

This command is used to place comments, remarks, hints or instructions in the report templates. These comments are not generated in the report.

Syntax

```
\Comments('<<Put your comment here. This will not be output in report>>')
```

Alternative Syntax

```
\C('<<Put your comment here. This will not be output in report>>')
```

Sample

```
\ Comments("The command below will fetch Actors from the current project ordered by Actor name")\
```

```
\Fetch_Actors_By_Condition ('','Name')\
```

```
\scan(a) \
```

```
\a:Name\ [\ a: Id\]
```

```
\endscan\
```

Customizing Templates in TopTeam

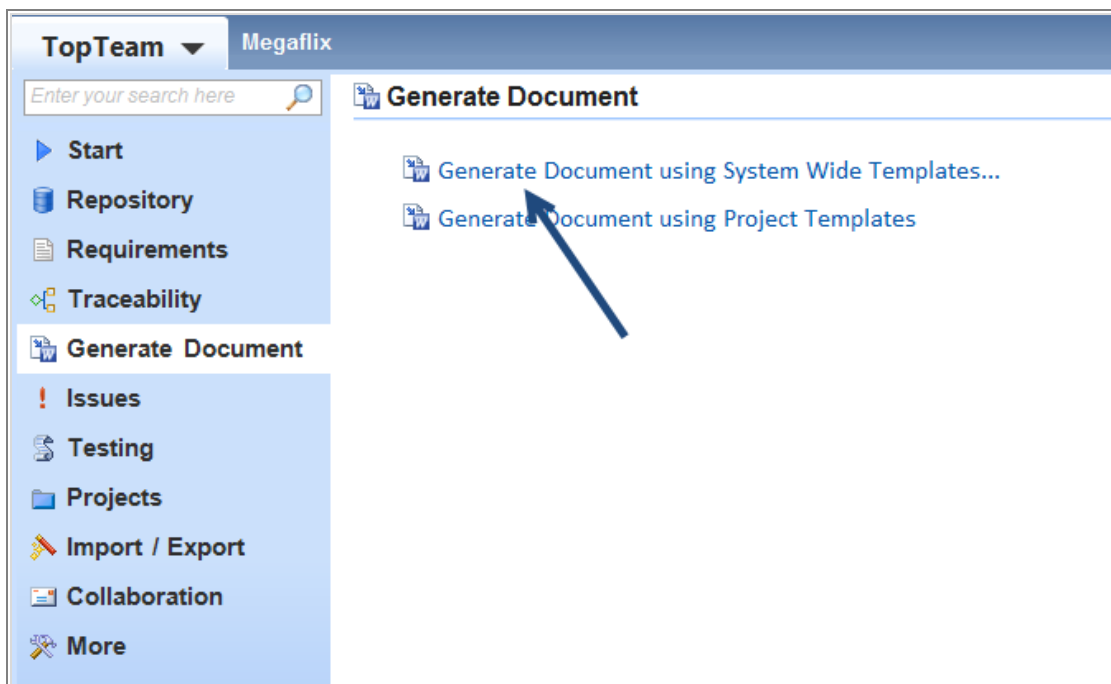
TopTeam ships with a number of default report templates which represent the most commonly used report layouts.

You can use these default templates as the starting point to create your own customized report layouts.

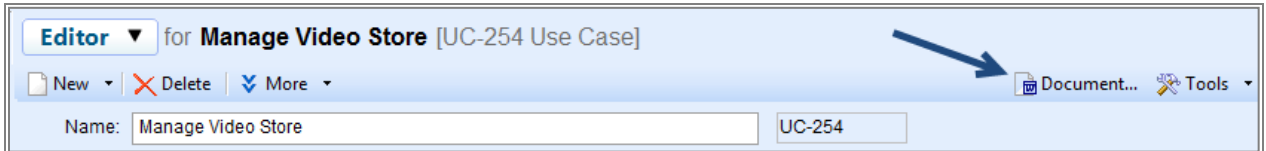
Template files bearing the name "Reference - All fields and all sub-reports" contain the majority of fields and sub-reports that can be placed in a report template.

All default templates are accessible from:

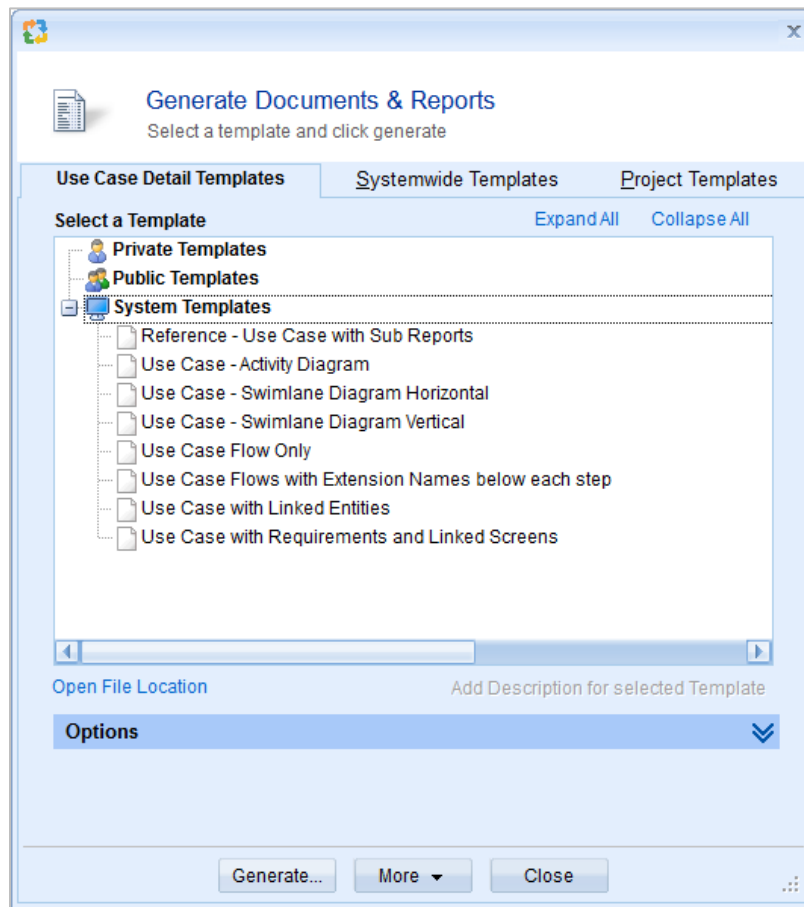
1. TopTeam main menu's Generate Document section, select option Generate Document using System Wide Templates...



2. The Document option on the toolbar of all editors.



Templates listed under the "System Templates" node are the default templates shipped with TopTeam.



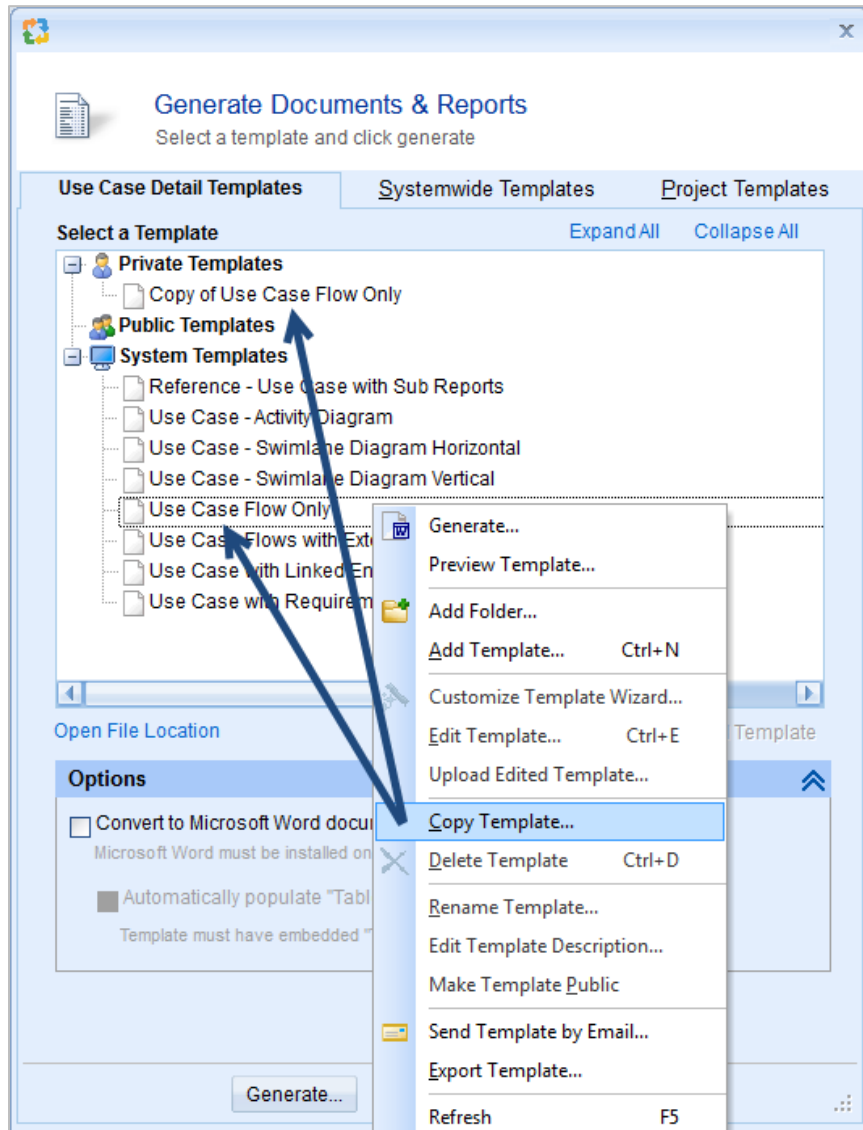
What is needed to customize a Report Template?

The Report Templates can be easily customized using a word processor such as Microsoft Word, etc. The word processor should be capable of editing Rich Text Format (*.rtf) files.

Modifying Templates

To define a new template, open an existing template that closely resembles the output you want.

Create a copy, save it with a new file name, and customize it.



How do I open Templates for customization?

Customize the existing template files by clicking the “Customize Template Wizard...” menu option.

This option will display the wizard which will allow you to:

1. Open the template for customization using the “Edit Template” option in the pop-up menu.
2. Modify the template manually, as required, and close the template file.
3. Load the template in the repository using the “Load Template” option in the pop-up menu.
4. You can also make a copy of the template and then customize it.

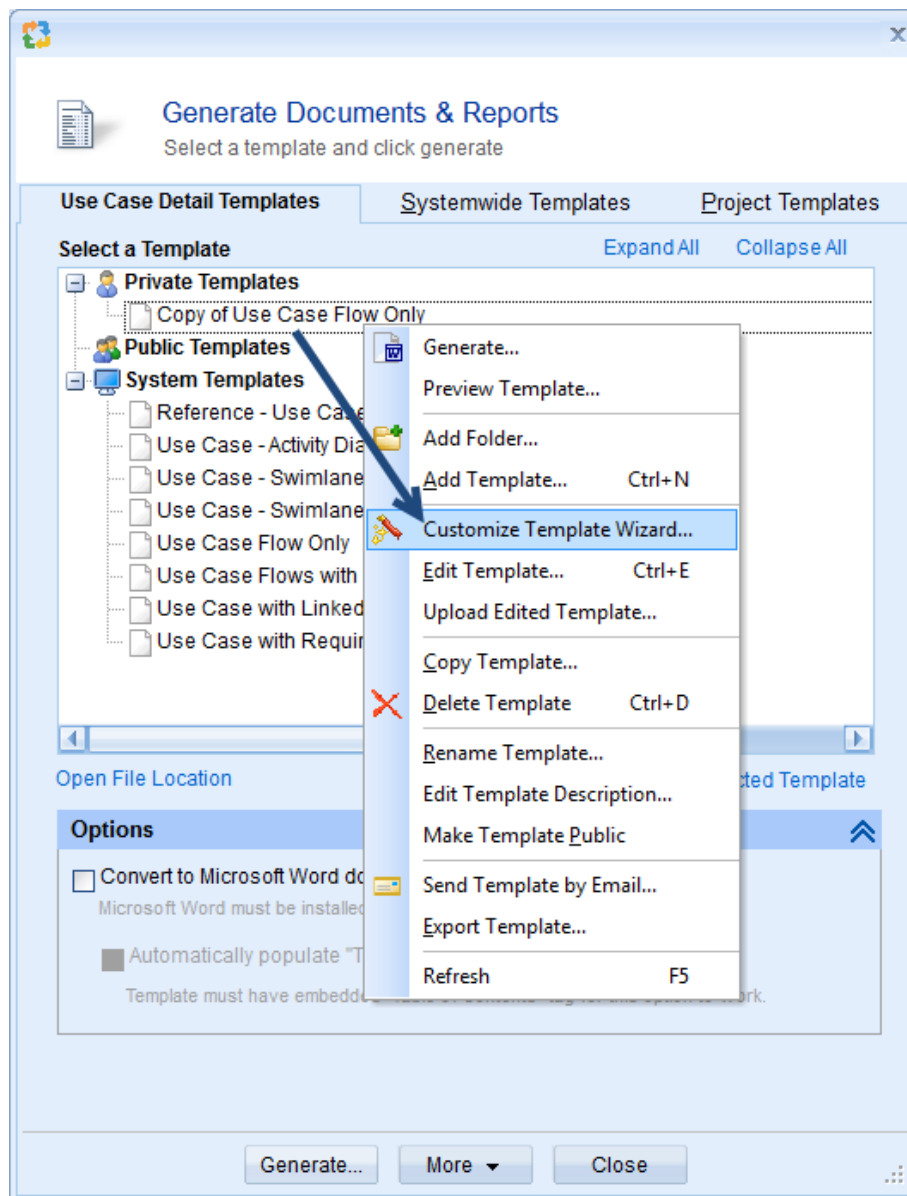
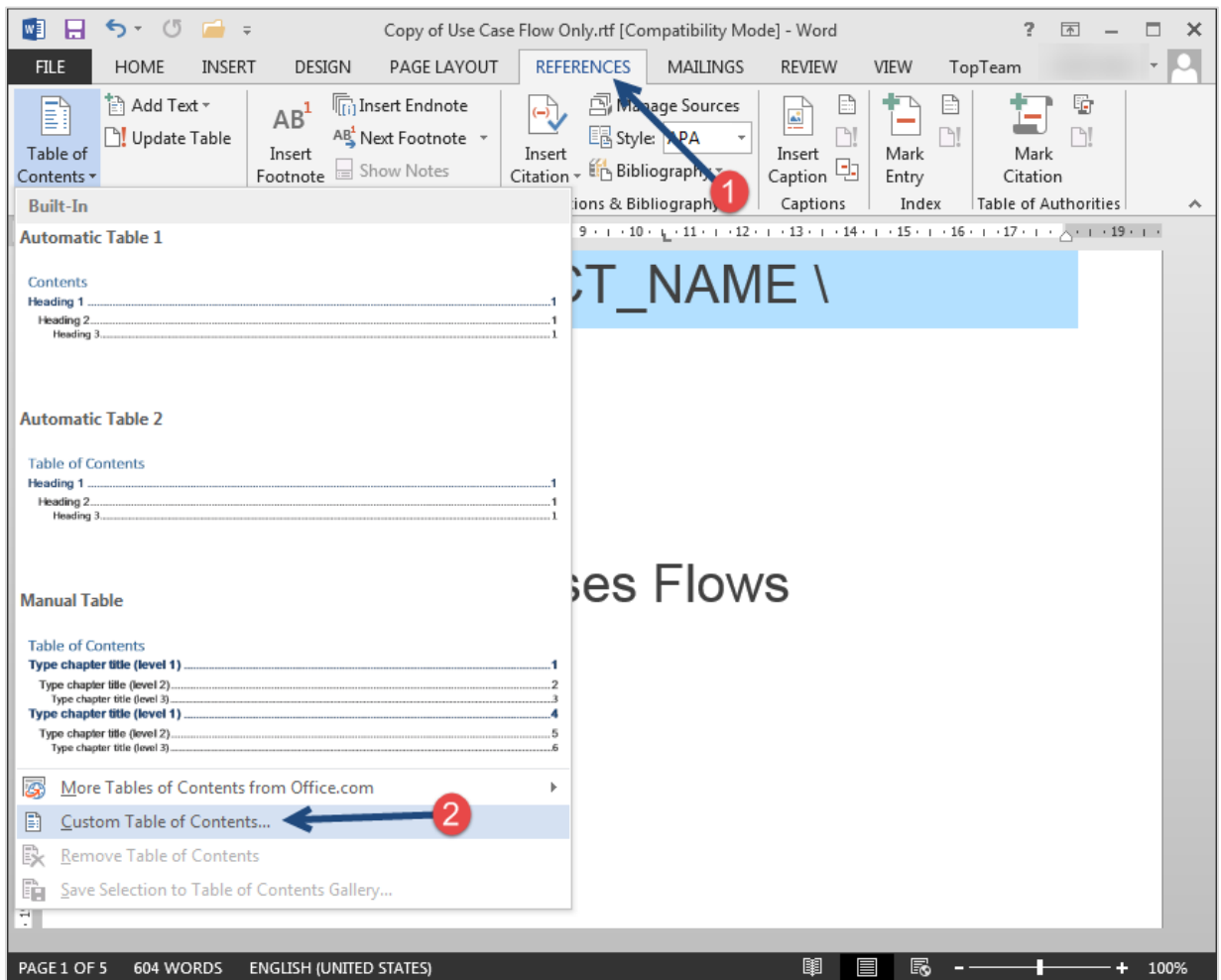


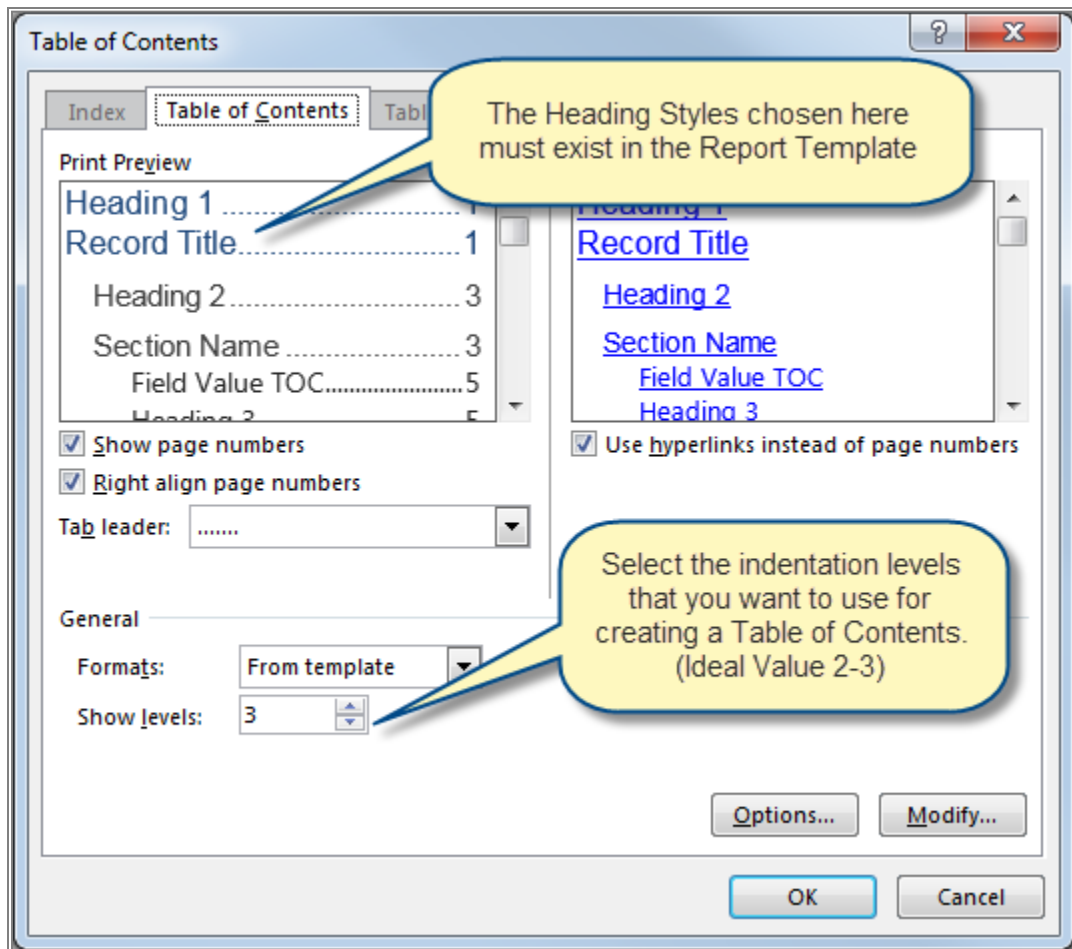
Table of Contents

A report template can be customized to generate a standard Table of Contents for the report. Perform the following steps to generate a Table of Contents:

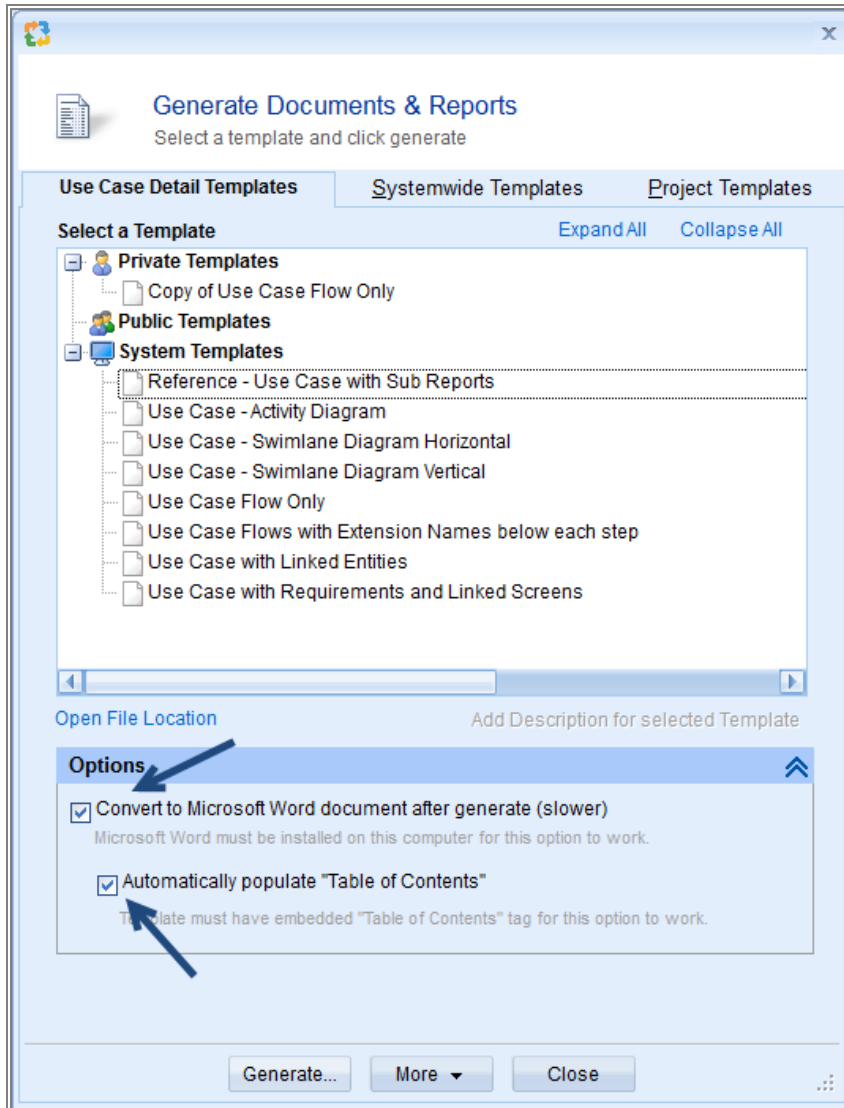
1. Open the template in Microsoft Word.
2. Place the cursor at the position where you want to place the Table of Contents.
3. Click the “Custom Table of Contents...” option, in the “Table of Contents” menu. (Office 2013 user interface shown below).



4. In the "Table of Contents" tab, choose the Heading Styles that you want to make Content Links in the report.



5. Convert to Microsoft Word document after generate (slower): Use the option to convert the report output (.rtf) into a Word document (.doc) or you can manually save the generated RTF report to a Word format (*.doc / *.docx).
6. Automatically populate Table of Contents: Use this option to update the Table of Contents, if a table of contents is already available in the report template.



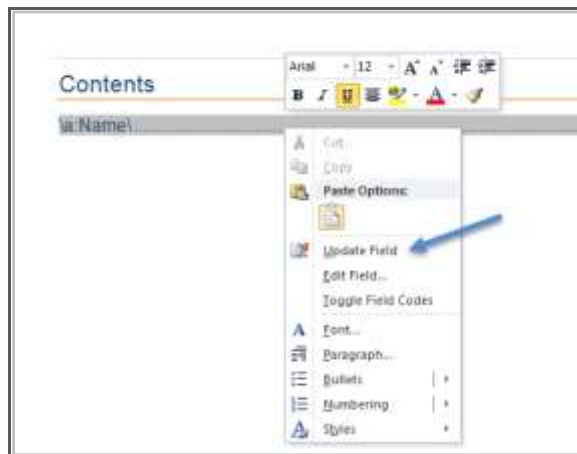
7. Upon generation, the Table of Contents will look like as shown below. Right-click and choose Update Field option from the pop-up menu to populate the Table of Contents.

Use Cases Flows	
Login to System	2
Main Flow-of-Events	2
Alternate Flows	2
3a. Customer forgot Credentials	2
4a. Incorrect credentials entered.....	2
Linked Requirements	3
Linked Screens Mockups.....	3

In the template, Use Case name is of Heading Style - Heading 1

Main Flow-of-Events, Alternate Flows, Linked Requirements and Linked Screen Mockups are of Heading Style - Heading 2

3a and 4a topics are of Heading Style - Heading 3



IMPORTANT

Microsoft Word must be installed on the local computer in order to generate the Table of Contents. If Microsoft Word is not installed on the computer on which the report is being generated, the Table Of Contents will not display. It will not affect the generated output as such. It will simply ignore the Table of Contents tags.

DocProcessor Limitations

- Templates created or edited using the "Open Office" word processor may not work correctly.
- When the report is viewed in the internal report viewer of the application, the Table of Contents is visible; however, the hyperlinks do not work.