Iterations

Iterating Issue History

Changes to issues are registered in the Issue History, but it is not known in advance how many changes are going to be made. You can iterate a section over all the history entries of an issue. This allows you to create a table that dynamically grows according to the number of changes dene. The notation is:

History Entry

Field	Description
HistoryEntriesCount	Returns the number of changes made.
Author	Returns the user who made the change.
Created	Date of the change
ChangedItemsCout	Returns the number of fields changed in the current change.
ChangedItem	**Field** / **Description** Field — Returns the name of the field which the value was changed. From — Returns the old value. To — Returns the new value.

Expand to see the sample code

```
#{for historyEntries}
   ${fullname:HistoryEntries[n].Author} made changes ${dateformat("dd-MM-yyyy HH:mm:ss"):HistoryEntries[n].
Created}
  #{for ch=HistoryEntries[n].ChangedItemsCount}
        Field Name: ${HistoryEntries[n].ChangedItems[ch].Field}
        Old Value:  ${wiki:HistoryEntries[n].ChangedItems[ch].From}
        New Value:  ${wiki:HistoryEntries[n].ChangedItems[ch].To}
  #{end}
#{end}

or

#{for h=HistoryEntriesCount}
   ${fullname:HistoryEntries[h].Author} made changes    ${dateformat("dd-MM-yyyy HH:mm:ss"):HistoryEntries[h].
Created}
   #{for ch=HistoryEntries[h].ChangedItemsCount}
        Field Name: ${HistoryEntries[h].ChangedItems[ch].Field}
        Old Value:  ${wiki:HistoryEntries[h].ChangedItems[ch].From}
        New Value:  ${wiki:HistoryEntries[h].ChangedItems[ch].To}
   #{end}
#{end}
```

Iterating Issue Activity

Changes to issues are registered in the Issue Activity, but it is not known in advance how many changes are going to be made. You can iterate a section over all the activities of an issue. This allows you to create a table that dynamically grows according to the number of existing activities. The notation is:

Activity Fields	Description
Title	The title of the issue
Summary	The summary of the activity
Content	When an activity involves a change in the Issue contents, this field displays the new contents
Author	The author of the activity
AuthorEmail	The email of the author of the activity
Published	The time the issue was published
Updated	The time the issue was updated
Categories	When an activity regards an Issue Status change, this field displays the new Issue Status

Expand to see the sample code

```
#{for activityEntries}
   ${ActivityEntries[n].Title}
   ${ActivityEntries[n].Summary}
   ${ActivityEntries[n].Content}
   ${ActivityEntries[n].Author}
   ${ActivityEntries[n].AuthorEmail}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[n].Published}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[n].Updated}
   ${ActivityEntries[n].Categories}
#{end}

or

#{for <VariableName>=ActivityEntriesCount}
   Content and Issue Mappings. Example: ${ActivityEntries[VariableName].Field}
#{end}
```

We suggest that you use the **html function** to render the data because almost all content is HTML, e.g., ${html:ActivityEntries[n].Title}

Below is an example of using the Activity iteration in a Word template:

#{for activityEntries}

Title	${ActivityEntries[n].Title}
Summary	${ActivityEntries[n].Summary}
Content	${ActivityEntries[n].Content}
Author	${ActivityEntries[n].Author}
Email	${ActivityEntries[n].AuthorEmail}
Published	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[n].Published}
Updated	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[n].Updated}
Categories	${ActivityEntries[n].Categories}

#{end}

or

#{for j=ActivityEntries}

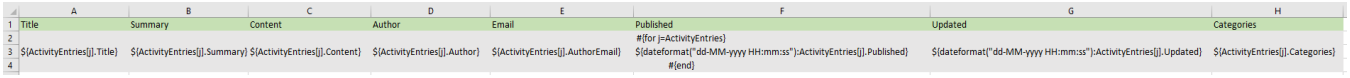
Title	${ActivityEntries[j].Title}
Summary	${ActivityEntries[j].Summary}
Content	${ActivityEntries[j].Content}
Author	${ActivityEntries[j].Author}
Email	${ActivityEntries[j].AuthorEmail}
Published	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[j].Published}
Updated	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[j].Updated}
Categories	${ActivityEntries[j].Categories}

#{end}

Below is an example of using the Activity iteration in an Excel template:

	A	B	C	D	E	F	G	H
1	Title	Summary	Content	Author	Email	Published	Updated	Categories
2						#{for activityEntries}		
3	${ActivityEntries[n].Title}	${ActivityEntries[n].Summary	${ActivityEntries[n].Content}	${ActivityEntries[n].Author}	${ActivityEntries[n].AuthorEmail}	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[n].Published}	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[n].Updated}	${ActivityEntries[n].Categories}
4						#{end}		

or



	A	B	C	D	E	F	G	H
1	Title	Summary	Content	Author	Email	Published	Updated	Categories
2						#{for j=ActivityEntries}		
3	${ActivityEntries[j].Title}	${ActivityEntries[j].Summary}	${ActivityEntries[j].Content}	${ActivityEntries[j].Author}	${ActivityEntries[j].AuthorEmail}	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[j].Published}	${dateformat("dd-MM-yyyy HH:mm:ss"):ActivityEntries[j].Updated}	${ActivityEntries[j].Categories}
4						#{end}		

Iterating Issue Links

Because it is not known in advance how many linked issues exist for an issue, you can iterate a section over all the linked issues of an issue. This allows you to create a table that dynamically grows according to the number of existing linked issues. The notation is:

Links Fields	Description
AppType	The application type of the link
LinkType	The type of the link
Key	The key of the linked issue
Summary	The summary of the linked issue
URL	The URL of the link

Expand to see the sample code

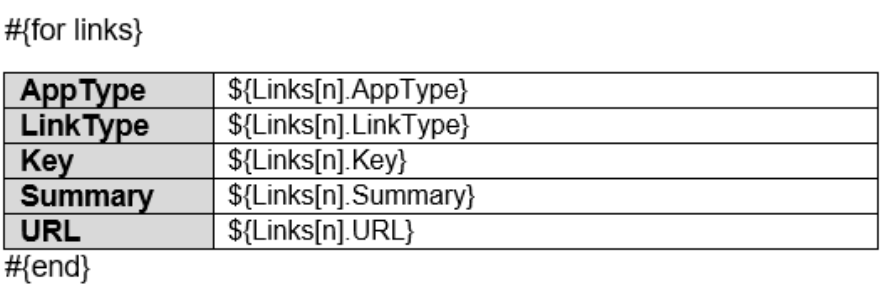
```
#{for links}
   ${Links[n].AppType}
   ${Links[n].LinkType}
   ${Links[n].Key}
   ${Links[n].Summary}
   ${Links[n].URL}
#{end}

or

#{for <VariableName>=LinksCount}
   Content and Linked Issue Mappings. Example: ${Links[VariableName].Field}
#{end}
```

All fields listed here are available on Links[n] because they represent an issue. In addition, there are two new fields at the Links[n] level:

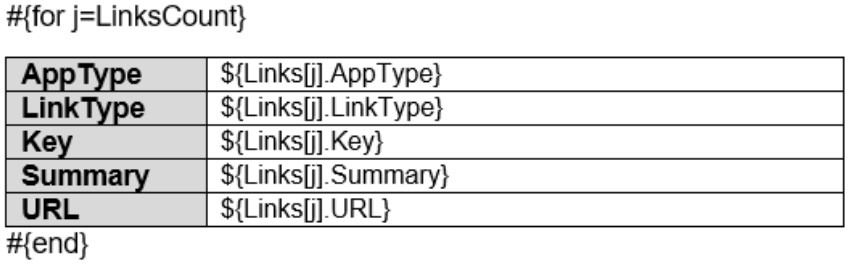
Field	Description
AppType	Returns the Application Type. The values can be: **Application Value** / **Description** **JIRA** — Link from the same Jira Instance **External Jira** — Link from the another Jira Instance **Confluence** — Link from a Confluence page **External** — External link
LinkType	Returns the Link Type.

Note: When the link you are iterating is of AppTypes **External Jira** or **Confluence,** the name is obtained using the Summary property.

The image below demonstrates an example of a Word template that iterates over linked issues.

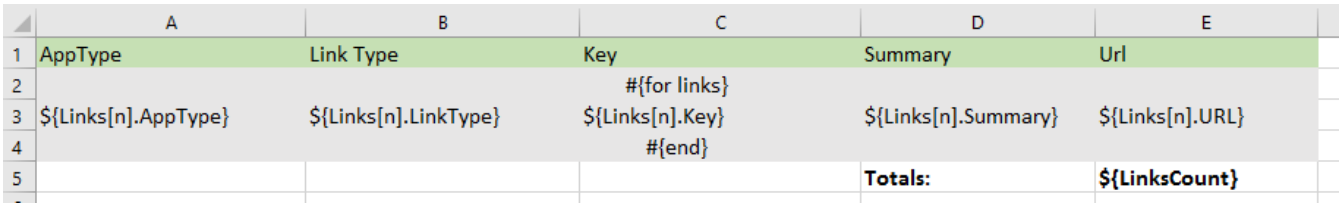


or



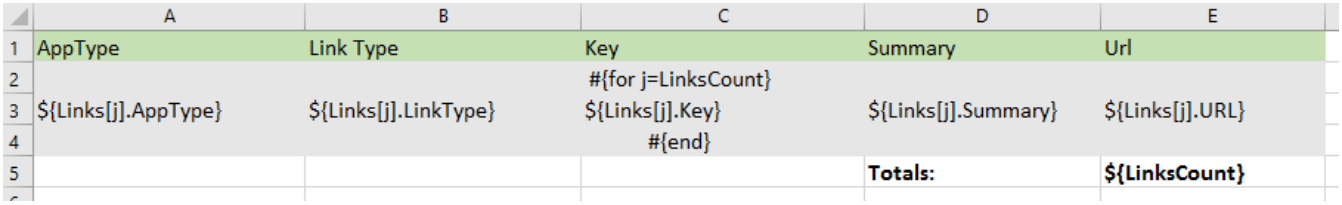
For a working example of this functionality, check the SampleIterations.docx template in the Template Store.

The image below demonstrates an example of an Excel template that iterates over linked issues.



	A	B	C	D	E
1	AppType	Link Type	Key	Summary	Url
2			#{for links}		
3	${Links[n].AppType}	${Links[n].LinkType}	${Links[n].Key}	${Links[n].Summary}	${Links[n].URL}
4			#{end}		
5				**Totals:**	**${LinksCount}**

or



	A	B	C	D	E
1	AppType	Link Type	Key	Summary	Url
2			#{for j=LinksCount}		
3	${Links[j].AppType}	${Links[j].LinkType}	${Links[j].Key}	${Links[j].Summary}	${Links[j].URL}
4			#{end}		
5				**Totals:**	**${LinksCount}**

Iterating Issue Comments

Because it is not known in advance how many comments exist for an issue, you can iterate a section over all the comments on an issue. This allows you to create a table that dynamically grows according to the number of existing comments. The notation is:

Comments Fields	Description
Author	The author of the comment
AuthorFullName	The full name of the author of the comment
Body	The comment
Created	The date the comment was posted
GroupLevel	The group level of the comment

Expand to see the sample code

```
#{for comments}
   ${Comments[n].Author}
   ${Comments[n].AuthorFullName}
   ${Comments[n].Body}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):Comments[n].Created}
   ${Comments[n].GroupLevel}
#{end}

or

#{for <VariableName>=CommentsCount}
   Content and Issue Mappings. Example: ${Comments[VariableName].Field}
#{end}
```

The image below demonstrates an example of a Word template that iterates over issue comments.

#{for comments}

Author	${Comments[n].Author}
AuthorFullName	${Comments[n].AuthorFullName}
Body	${Comments[n].Body}
Creation date	${dateformat("dd-MM-yyyy"):Comments[n].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Comments[n].Created}
GroupLevel	${Comments[n].GroupLevel}

#{end}

or

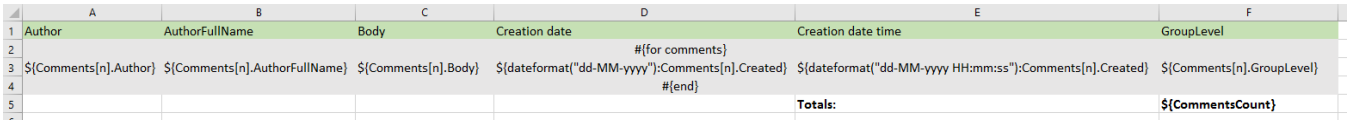
#{for j=CommentsCount}

Author	${Comments[j].Author}
AuthorFullName	${Comments[j].AuthorFullName}
Body	${Comments[j].Body}
Creation date	${dateformat("dd-MM-yyyy"):Comments[j].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Comments[j].Created}
GroupLevel	${Comments[j].GroupLevel}

#{end}

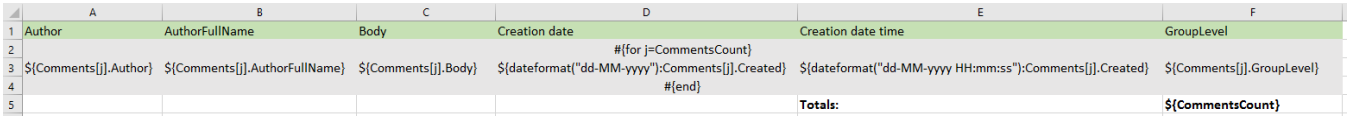
For a working example of this functionality, check the SampleIterations.docx template in the Template Store.

The image below demonstrates an example of an Excel template that iterates over issue comments.



	A	B	C	D	E	F
1	Author	AuthorFullName	Body	Creation date	Creation date time	GroupLevel
2				#{for comments}		
3	${Comments[n].Author}	${Comments[n].AuthorFullName}	${Comments[n].Body}	${dateformat("dd-MM-yyyy"):Comments[n].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Comments[n].Created}	${Comments[n].GroupLevel}
4				#{end}		
5					**Totals:**	**${CommentsCount}**

or



	A	B	C	D	E	F
1	Author	AuthorFullName	Body	Creation date	Creation date time	GroupLevel
2				#{for j=CommentsCount}		
3	${Comments[j].Author}	${Comments[j].AuthorFullName}	${Comments[j].Body}	${dateformat("dd-MM-yyyy"):Comments[j].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Comments[j].Created}	${Comments[j].GroupLevel}
4				#{end}		
5					**Totals:**	**${CommentsCount}**

Jira Service Desk

If you are using Jira Service Desk you can see more information about comments here.

Iterating Issue Worklogs

Because it is not known in advance how many worklogs exist for an issue, you can iterate a section over all the worklogs of an issue. This allow you to create a table that dynamically grows according to the number of existing worklogs. The notation is:

Worklogs Fields	Description
Author	The author of the worklog
AuthorFullName	The full name of the author of the worklog
Comment	The comment of the worklog
Created	The date the worklog was created
Date Started	The date the worklog was started
Time Spent	The time spent in seconds
TimeSpentFormatted	The time spent as displayed on Jira
BilledHours	The billed hours in seconds (**Belongs to Tempo Timesheets plugin**)
BilledHoursFormatted	The billed hours as displayed on Jira (**Belongs to Tempo Timesheets plugin**)

Expand to see the sample code

```
#{for worklogs}
   ${Worklogs[n].Author}
   ${Worklogs[n].AuthorFullName}
   ${Worklogs[n].Comment}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[n].Created}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[n].Date Started}
   ${Worklogs[n].Time Spent}
   ${Worklogs[n].TimeSpentFormatted}
   ${Worklogs[n].BilledHours}
   ${Worklogs[n].BilledHoursFormatted}
#{end}

or

#{for <VariableName>=WorklogsCount}
   Content and Worklog Mappings. Example: ${Worklogs[VariableName].Field}
#{end}
```

The image below demonstrates an example of a Word template that iterates over issue worklogs.

#{for worklogs}

Author	${Worklogs[n].Author}
AuthorFullName	${Worklogs[n].AuthorFullName}
Comment	${Worklogs[n].Comment}
Creation date	${dateformat("dd-MM-yyyy"):Worklogs[n].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[n].Created}
Date Started	${dateformat("dd-MM-yyyy"):Worklogs[n].Date Started}
Date Started time	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[n].Date Started}
Time Spent	${Worklogs[n].Time Spent}
TimeSpentFormatted	${Worklogs[n].TimeSpentFormatted}
BilledHours	${Worklogs[n].BilledHours}
BilledHoursFormatted	${Worklogs[n].BilledHoursFormatted}

#{end}

or

#{for j=WorklogsCount}

Author	${Worklogs[j].Author}
AuthorFullName	${Worklogs[j].AuthorFullName}
Comment	${Worklogs[j].Comment}
Creation date	${dateformat("dd-MM-yyyy"):Worklogs[j].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[j].Created}
Date Started	${dateformat("dd-MM-yyyy"):Worklogs[j].Date Started}
Date Started time	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[j].Date Started}
Time Spent	${Worklogs[j].Time Spent}
TimeSpentFormatted	${Worklogs[j].TimeSpentFormatted}
BilledHours	${Worklogs[j].BilledHours}
BilledHoursFormatted	${Worklogs[j].BilledHoursFormatted}

#{end}

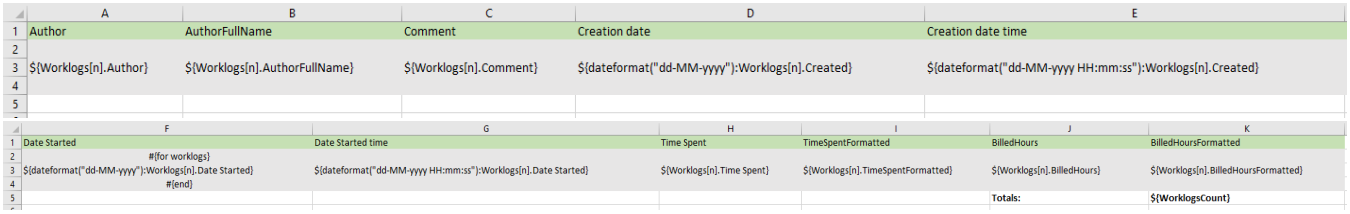
For a working example of this functionality, check the SampleIterations.docx template in the Template Store.

The image below demonstrates an example of a template in Excel that iterates over issue work logs.

	A	B	C	D	E
1	Author	AuthorFullName	Comment	Creation date	Creation date time
2					
3	${Worklogs[j].Author}	${Worklogs[j].AuthorFullName}	${Worklogs[j].Comment}	${dateformat("dd-MM-yyyy"):Worklogs[j].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[j].Created}
4					
5					

	F	G	H	I	J	K
1	Date Started	Date Started time	Time Spent	TimeSpentFormatted	BilledHours	BilledHoursFormatted
2	#{for j=WorklogsCount}					
3	${dateformat("dd-MM-yyyy"):Worklogs[j].Date Started}	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[j].Date Started}	${Worklogs[j].Time Spent}	${Worklogs[j].TimeSpentFormatted}	${Worklogs[j].BilledHours}	${Worklogs[j].BilledHoursFormatted}
4	#{end}					
5					**Totals:**	**${WorklogsCount}**

or



	A	B	C	D	E
1	Author	AuthorFullName	Comment	Creation date	Creation date time
2					
3	${Worklogs[n].Author}	${Worklogs[n].AuthorFullName}	${Worklogs[n].Comment}	${dateformat("dd-MM-yyyy"):Worklogs[n].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[n].Created}
4					
5					

	F	G	H	I	J	K
1	Date Started	Date Started time	Time Spent	TimeSpentFormatted	BilledHours	BilledHoursFormatted
2	#{for worklogs}					
3	${dateformat("dd-MM-yyyy"):Worklogs[n].Date Started}	${dateformat("dd-MM-yyyy HH:mm:ss"):Worklogs[n].Date Started}	${Worklogs[n].Time Spent}	${Worklogs[n].TimeSpentFormatted}	${Worklogs[n].BilledHours}	${Worklogs[n].BilledHoursFormatted}
4	#{end}					
5					**Totals:**	**${WorklogsCount}**

Iterating Issue Subtasks

Because it is not known in advance how many subtasks exist for an issue, you can iterate a section over all the subtasks of an issue. This allows you to create a table that dynamically grows according to the number of existing subtasks. The notation is:

Subtasks Fields	Description
Key	The key of the subtasks
Summary	The summary of the subtasks
AssigneeUserDisplayName	The assignee user of the subtasks

Expand to see the sample code

```
#{for subtasks}
   ${Subtasks[n].Key}
   ${Subtasks[n].Summary}
   ${Subtasks[n].AssigneeUserDisplayName}
#{end}

or

#{for <VariableName>=SubtasksCount}
   Content and Issue Mappings. Example: ${Subtasks[VariableName].Field}
#{end}
```

The image below demonstrates an example of a Word template that iterates over issue subtasks.

#{for subtasks}

Key	${Subtasks[n].Key}
Summary	${Subtasks[n].Summary}
AssigneeUserDisplayName	${Subtasks[n].AssigneeUserDisplayName}

#{end}

or

#{for j=SubtasksCount}

Key	${Subtasks[j].Key}
Summary	${Subtasks[j].Summary}
AssigneeUserDisplayName	${Subtasks[j].AssigneeUserDisplayName}

#{end}

For a working example of this functionality, check the SampleIterations.docx template in the Template Store.

The image below demonstrates an example of an Excel template that iterates over issue subtasks.

	A	B	C
1	Key	Summary	Assignee
2		#{for subtasks}	
3	${Subtasks[n].Key}	${Subtasks[n].Summary}	${Subtasks[n].AssigneeUserDisplayName}
4		#{end}	
5		**Totals:**	**${SubtasksCount}**

or

	A	B	C
1	Key	Summary	Assignee
2		#{for j=SubtasksCount}	
3	${Subtasks[j].Key}	${Subtasks[j].Summary}	${Subtasks[j].AssigneeUserDisplayName}
4		#{end}	
5		**Totals:**	**${SubtasksCount}**

For an example of how to iterate the details of a subtask Parent issue, please check the Iterating JQL Queries area below.

Iterating Issue Components

Because it is not known in advance how many components exist for an issue, you can iterate a section over all the components of an issue. This allows you to create a table that dynamically grows according to the number of existing components. The notation is:

Components Fields	Description
Name	The name of the component
Description	The description of the component
Lead	The name of the component lead
Id	The ID of the component
ProjectId	The project ID of the component
AssigneeType	The assignee type of the component

Expand to see the sample code

```
#{for components}
   ${Components[n].Name}
   ${Components[n].Description}
   ${fullname:Components[n].Lead}
   ${Components[n].Id}
   ${Components[n].ProjectId}
   ${Components[n].AssigneeType}
#{end}
```

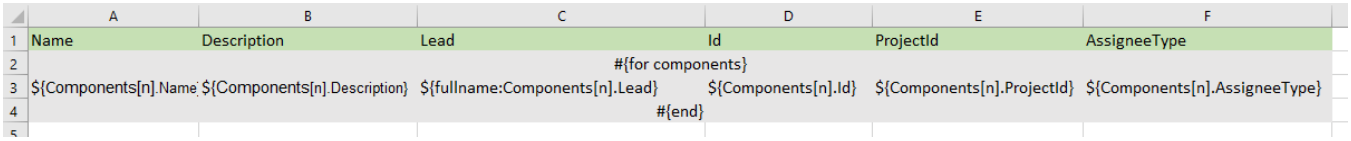
The image below demonstrates an example of a Word template that iterates over issue components.

#{for components}

Name	${Components[n].Name}
Description	${Components[n].Description}
Lead	${fullname:Components[n].Lead}
Id	${Components[n].Id}
ProjectId	${Components[n].ProjectId}
AssigneeType	${Components[n].AssigneeType}

#{end}

The image below demonstrates an example of an Excel template that iterates over issue components.



	A	B	C	D	E	F
1	Name	Description	Lead	Id	ProjectId	AssigneeType
2			#{for components}			
3	${Components[n].Name}	${Components[n].Description}	${fullname:Components[n].Lead}	${Components[n].Id}	${Components[n].ProjectId}	${Components[n].AssigneeType}
4			#{end}			

Iterating Issue Status Transitions

Because it is not known in advance how many Status Transitions exist for an issue, you can iterate a section over all the Status Transitions of an issue. This allows you to create a table that dynamically grows according to the number of existing status transitions. The notation is:

Status Transitions Fields	Description
Author	The author of the status transition
Created	The date the status transition was performed
OldStatus	The old status of the status transition
NewStatus	The new status of the status transition

Expand to see the sample code

```
#{for statusTransitions}
   ${StatusTransitions[n].Author}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):StatusTransitions[n].Created}
   ${StatusTransitions[n].OldStatus}
   ${StatusTransitions[n].NewStatus}
#{end}

or

#{for <VariableName>=StatusTransitionsCount}
   Content and StatusTransitions Mappings. Example: ${StatusTransitions[VariableName].Field}
#{end}
```

The image below demonstrates an example of a Word template that iterates over status transitions.

#{for statusTransitions}

Author	${StatusTransitions[n].Author}
Creation date	${dateformat("dd-MM-yyyy"):StatusTransitions[n].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):StatusTransitions[n].Created}
OldStatus	${StatusTransitions[n].OldStatus}
NewStatus	${StatusTransitions[n].NewStatus}

#{end}

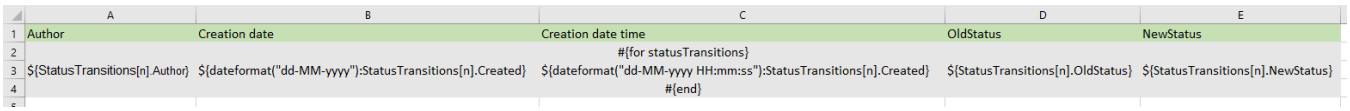
or

#{for j=StatusTransitionsCount}

Author	${StatusTransitions[j].Author}
Creation date	${dateformat("dd-MM-yyyy"):StatusTransitions[j].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):StatusTransitions[j].Created}
OldStatus	${StatusTransitions[j].OldStatus}
NewStatus	${StatusTransitions[j].NewStatus}

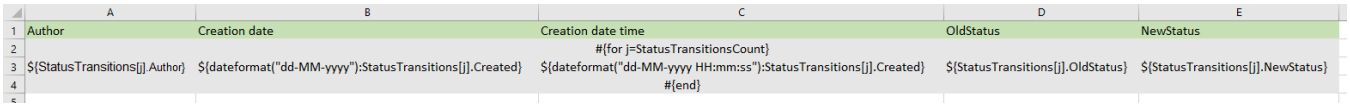
#{end}

The image below demonstrates an example of an Excel template that iterates over status transitions.



	A	B	C	D	E
1	Author	Creation date	Creation date time	OldStatus	NewStatus
2			#{for statusTransitions}		
3	${StatusTransitions[n].Author}	${dateformat("dd-MM-yyyy"):StatusTransitions[n].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):StatusTransitions[n].Created}	${StatusTransitions[n].OldStatus}	${StatusTransitions[n].NewStatus}
4			#{end}		

or



	A	B	C	D	E
1	Author	Creation date	Creation date time	OldStatus	NewStatus
2			#{for j=StatusTransitionsCount}		
3	${StatusTransitions[j].Author}	${dateformat("dd-MM-yyyy"):StatusTransitions[j].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):StatusTransitions[j].Created}	${StatusTransitions[j].OldStatus}	${StatusTransitions[j].NewStatus}
4			#{end}		

Iterating Issue Attached Images

Because it is not known in advance how many Images can exist for an issue (as an attachment), you can iterate a section over all the attached images of an issue to get some metadata about them. This allows you to create a table that dynamically grows according to the number of existing images. The notation is:

Attachments Images Fields	Description
ID	The ID of the attached image
Image	The image of the attached image
Name	The name of the attached image
Size	The size of the attached image
HumanReadableSize	The size of the attached image
Author	The author of the attached image
Created	The date the attached image was created
MimeType	The type of the attached image
ThumbnailURL	The URL to the thumbnail of the image

Expand to see the sample code

```
#{for images}
   ${Images[n].Image|maxwidth=150|maxheight=150}
   ${Images[n].Name}
   ${Images[n].ID}
   ${Images[n].Size}
   ${Images[n].HumanReadableSize}
   ${Images[n].Author}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):Images[n].Created}
   ${Images[n].MimeType}
   ${Images[n].ThumbnailURL}
 #{end}

or

#{for <VariableName>=ImagesCount}
   Content and Images Mappings. Example: ${Images[VariableName].Field}
#{end}
```

The image below demonstrates an example of a Word template that iterates over attached images.

#{for images}

${Images[n].Image|maxwidth=150|maxheight=150}

Name	${Images[n].Name}
ID	${Images[n].ID}
Size	${Images[n].Size}
HumanReadableSize	${Images[n].HumanReadableSize}
Author	${Images[n].Author}
Creation date	${dateformat("dd-MM-yyyy"):Images[n].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Images[n].Created}
MimeType	${Images[n].MimeType}

#{end}

or

#{for j=ImagesCount}

${Images[j].Image|maxwidth=150|maxheight=150}

Name	${Images[j].Name}
ID	${Images[j].ID}
Size	${Images[j].Size}
HumanReadableSize	${Images[j].HumanReadableSize}
Author	${Images[j].Author}
Creation date	${dateformat("dd-MM-yyyy"):Images[j].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Images[j].Created}
MimeType	${Images[j].MimeType}

#{end}

Xporter will automatically read the EXIF orientation property of an image and rotate it to its correct orientation. You can turn this off by adding this property to your template.

Since Xporter 5.5.0, you can use the mappings width and height to define the exact width and height of the printed image.

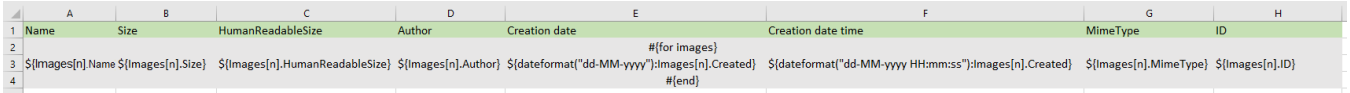
Expand to see the sample code

```
#{for images}
   ${Images[n].Image|width=150|height=150}
 #{end}
```

These values are in pixels and if you only define one of them the image will be rescaled.

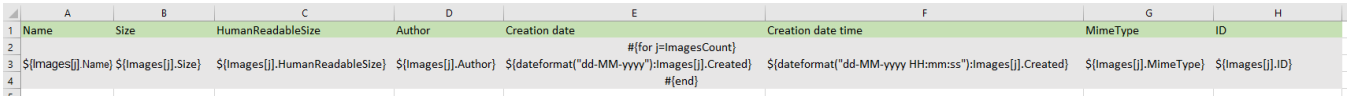
Note that, if you use both maxWidth and width mappings, only the max value will be read. The same behavior happens with height and maxHeight.

The image below demonstrates an example of an Excel template that iterates over attached images.



	A	B	C	D	E	F	G	H
1	Name	Size	HumanReadableSize	Author	Creation date	Creation date time	MimeType	ID
2					#{for images}			
3	${Images[n].Name	${Images[n].Size}	${Images[n].HumanReadableSize}	${Images[n].Author}	${dateformat("dd-MM-yyyy"):Images[n].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Images[n].Created}	${Images[n].MimeType}	${Images[n].ID}
4					#{end}			

or



	A	B	C	D	E	F	G	H
1	Name	Size	HumanReadableSize	Author	Creation date	Creation date time	MimeType	ID
2					#{for j=ImagesCount}			
3	${Images[j].Name}	${Images[j].Size}	${Images[j].HumanReadableSize}	${Images[j].Author}	${dateformat("dd-MM-yyyy"):Images[j].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Images[j].Created}	${Images[j].MimeType}	${Images[j].ID}
4					#{end}			

You can iterate over ${Images[n].Image} in Excel on Xporter version 5.4.0 and above.

Iterating Issue Attachments

Because it is not known in advance how many attachments exist in an issue, you can iterate a section over all the attachments of an issue. This allows you to create a table that dynamically grows according to the number of existing attachments. The notation is:

Attachments Fields	Description
ID	The ID of the attachment
Name	The name of the attachment
Author	The author of the attachment
AuthorFullName	The full name of the author of the attachment
Created	The date the attachment was created
Size	The size of the attachment
HumanReadableSize	The formatted size of the attachment
MimeType	The type of the attachment

Expand to see the sample code

```
#{for attachments}
   ${Attachments[n].ID}
   ${Attachments[n].Name}
   ${Attachments[n].Author}
   ${Attachments[n].AuthorFullName}
   ${dateformat("dd-MM-yyyy HH:mm:ss"):Attachments[n].Created}
   ${Attachments[n].Size}
   ${Attachments[n].HumanReadableSize}
   ${Attachments[n].MimeType}
#{end}

or

#{for <VariableName>=AttachmentsCount}
   Content and Issue Mappings. Example: ${Attachments[VariableName].Field}
#{end}
```

The image below demonstrates an example of a Word template that iterates over attachments.

#{for attachments}

ID	${Attachments[n].ID}
Name	${Attachments[n].Name}
Author	${Attachments[n].Author}
AuthorFullName	${Attachments[n].AuthorFullName}
Creation date	${dateformat("dd-MM-yyyy"):Attachments[n].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Attachments[n].Created}
Size	${Attachments[n].Size}
HumanReadableSize	${Attachments[n].HumanReadableSize}
MimeType	${Attachments[n].MimeType}

#{end}

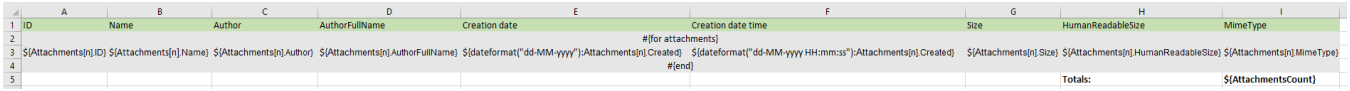
or

#{for j=AttachmentsCount}

ID	${Attachments[j].ID}
Name	${Attachments[j].Name}
Author	${Attachments[j].Author}
AuthorFullName	${Attachments[j].AuthorFullName}
Creation date	${dateformat("dd-MM-yyyy"):Attachments[j].Created}
Creation date time	${dateformat("dd-MM-yyyy HH:mm:ss"):Attachments[j].Created}
Size	${Attachments[j].Size}
HumanReadableSize	${Attachments[j].HumanReadableSize}
MimeType	${Attachments[j].MimeType}

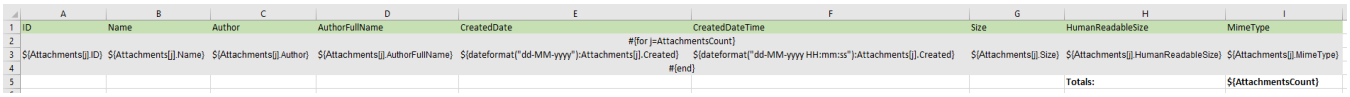
#{end}

The image below demonstrates an example of an Excel template that iterates over attachments.



	A	B	C	D	E	F	G	H	I
1	ID	Name	Author	AuthorFullName	Creation date	Creation date time	Size	HumanReadableSize	MimeType
2					#{for attachments}				
3	${Attachments[n].ID}	${Attachments[n].Name}	${Attachments[n].Author}	${Attachments[n].AuthorFullName}	${dateformat("dd-MM-yyyy"):Attachments[n].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Attachments[n].Created}	${Attachments[n].Size}	${Attachments[n].HumanReadableSize}	${Attachments[n].MimeType}
4					#{end}				
5								**Totals:**	**${AttachmentsCount}**

or



	A	B	C	D	E	F	G	H	I
1	ID	Name	Author	AuthorFullName	CreatedDate	CreatedDateTime	Size	HumanReadableSize	MimeType
2					#{for j=AttachmentsCount}				
3	${Attachments[j].ID}	${Attachments[j].Name}	${Attachments[j].Author}	${Attachments[j].AuthorFullName}	${dateformat("dd-MM-yyyy"):Attachments[j].Created}	${dateformat("dd-MM-yyyy HH:mm:ss"):Attachments[j].Created}	${Attachments[j].Size}	${Attachments[j].HumanReadableSize}	${Attachments[j].MimeType}
4					#{end}				
5								**Totals:**	**${AttachmentsCount}**

Iterating Issue Labels

Because it is not known in advance how many labels exist in an issue, you can iterate a section over all the labels of an issue. The notation is:

Attachments Fields	Description
Name	The name of the label

```
#{for labels}
   ${Labels[n].Name}
#{end}

or

#{for <VariableName>=LabelsCount}
  ${Labels[VariableName].Name}
#{end}
```

The image below demonstrates an example of a Word template that iterates over labels.

#{for labels}

Name	${Labels[n].Name}

#{end}

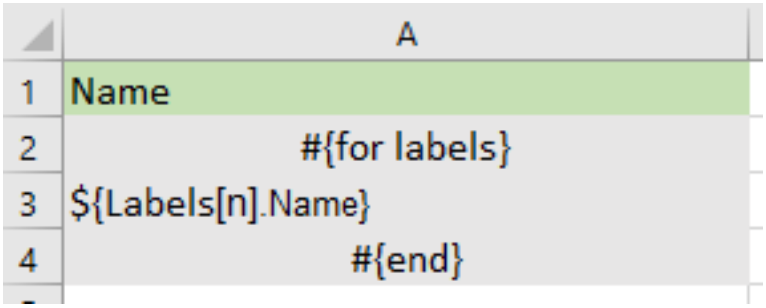
or

#{for f=LabelsCount}

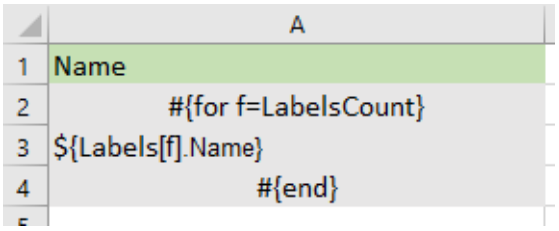
Name	${Labels[f].Name}

#{end}

The image below demonstrates an example of an Excel template that iterates over labels.



or



Iterating Project Versions from an Issue

You can iterate over all project versions to which the issue belong to. The notation is:

Attachments Fields	Description
Name	The name of the project version
Description	The description of the project version
`Start date`	The Start Date of the project version
`Release date`	The Release Date of the project version

```
#{for projectVersions}
   ${ProjectVersions[n].Name}
   ${ProjectVersions[n].Description}
   ${dateformat("dd-MM-yyyy"):ProjectVersions[n].Start date}
   ${dateformat("dd-MM-yyyy"):ProjectVersions[n].Release date}
#{end}

or

#{for <VariableName>=ProjectVersionsCount}
   ${ProjectVersions[VariableName].Name}
   ${ProjectVersions[VariableName].Description}
   ${dateformat("dd-MM-yyyy"):ProjectVersions[VariableName].Start date}
   ${dateformat("dd-MM-yyyy"):ProjectVersions[VariableName].Release date}
#{end}
```

The image below demonstrates an example of a Word template that iterates over project version.

#{for projectVersions}

Name	${ProjectVersions[n].Name}
Description	${ProjectVersions[n].Description}
Start date	${dateformat("yyyy-MM-dd"):ProjectVersions[n].Start date}
Release date	${dateformat("yyyy-MM-dd"):ProjectVersions[n].Release date}

#{end}

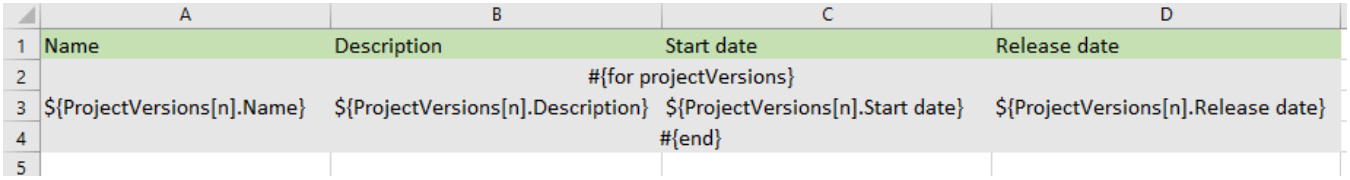
or

#{for j=ProjectVersionsCount }

Name	${ProjectVersions[j].Name}
Description	${ProjectVersions[j].Description}
Start date	${dateformat("yyyy-MM-dd"):ProjectVersions[j].Start date}
Release date	${dateformat("yyyy-MM-dd"):ProjectVersions[j].Release date}

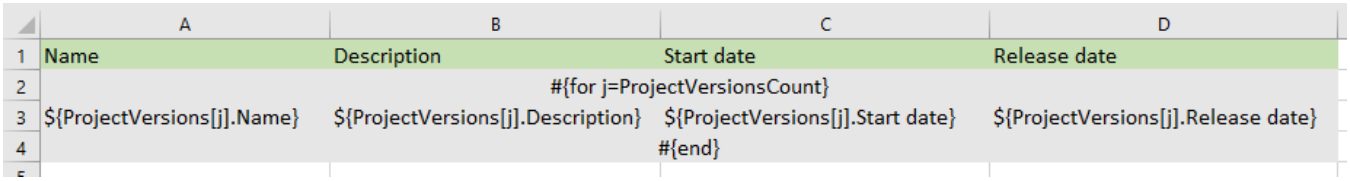
#{end}

The image below demonstrates an example of an Excel template that iterates over project version.



	A	B	C	D
1	Name	Description	Start date	Release date
2			#{for projectVersions}	
3	${ProjectVersions[n].Name}	${ProjectVersions[n].Description}	${ProjectVersions[n].Start date}	${ProjectVersions[n].Release date}
4			#{end}	
5				

or



	A	B	C	D
1	Name	Description	Start date	Release date
2			#{for j=ProjectVersionsCount}	
3	${ProjectVersions[j].Name}	${ProjectVersions[j].Description}	${ProjectVersions[j].Start date}	${ProjectVersions[j].Release date}
4			#{end}	

Iterating JQL Queries

You can iterate issues that are the result of a JQL Query. The syntax is similar to the other iterations, but there is a **clause** parameter that will receive the JQL Query. A few examples are provided below.

Expand to see the sample code

```
a simple example iterating the details of issues from a specified Project:

#{for i=JQLIssuesCount|clause=project = DEMO}
   ${JQLIssues[i].Key}
   ${JQLIssues[i].Summary}
#{end}

or a more advanced example iterating the details of issues linked with the current Issue:

#{for m=JQLIssuesCount|clause=issuekey in linkedIssues (${Links[j].Key})}
   Linked Issue ${JQLIssues[m].Summary} has ${JQLIssues[m].LinksCount} links
#{end}

or an also advanced example iterating the details of the Parent issue from the current Subtask:

#{for i=JQLIssuesCount|clause=issuekey = ${ParentIssueKey}}
        ${JQLIssues[i].Key}
        ${JQLIssues[i].Id}
        ${JQLIssues[i].Description}
#{end}
```

The image below demonstrates an example of a Word template that iterates over issue subtasks.

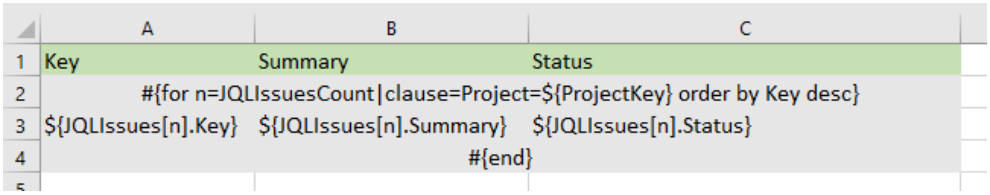
#{for n=JQLIssuesCount|clause=Project=${ProjectKey} order by Key desc}

Key	${JQLIssues[n].Key}
Summary	${JQLIssues[n].Summary}
Status	${JQLIssues[n].Status}

#{end}

For a working example of this functionality, check the template Sample Iterations in the Template Store.

The image below demonstrates an example of an Excel template that iterates over issue subtasks.



You can also use a Filter Name or a Filter Id as a clause. For more info, check [http://confluence.xpand-addons.com/display/public/XPORTER/JQL]

Iterating Issue Commits

Because it is not known in advance how many commits exist for an issue, you can iterate a section over all the commits of an issue. This allows you to create a table that dynamically grows according to the number of existing commits. The notation is:

Commits Fields	Description
URL	The URL of the link
CreatedDateTime	The date the commit was created
`Message`	The message of the commit
`Author`	The author of the commit

In order to extract more information from Commits, it is possible to get information from the files that were committed:

Commits files count Fields	Description
FilesCount.Path	The path of the file was committed
FilesCount.URL	The URL of the file was committed
FilesCount.ChangeType	Identify the type of change that occurred in the commit

#{for commits}

URL	${Commits[n].URL}
CreatedDateTime	${Commits[n].CreatedDateTime}
Message	${Commits[n].Message}
Author	${Commits[n].Author}

Nested Iterations:

#{for m=Commits[n].FilesCount}

Path	${Commits[n].FilesCount[m].Path}
URL	${Commits[n].FilesCount[m].URL}
ChangeType	${Commits[n].FilesCount[m].ChangeType}

#{end}

#{end}

Expand to see the sample code

```
#{for commits}
        ${Commits[n].Author}
        ${Commits[n].URL}
        ${Commits[n].Message}
        ${Commits[n].CreatedDateTime}

        Here we have the FilesCount where we can get all the files associated with a commit.
        #{for m=Commits[n].FilesCount}
                ${Commit[n].FilesCount[m].Path}
                ${Commit[n].FilesCount[m].URL}
                ${Commit[n].FilesCount[m].ChangeType}
        #{end}
#{end}

or

#{for <VariableName>=CommitsCount}
   Content and Issue Mappings. Example: ${Commits[VariableName].Field}
#{end}
```

Iterating Issue Branches

Because it is not known in advance how many branches exist for an issue, you can iterate a section over all the branches of an issue. This allows you to create a table that dynamically grows according to the number of existing branches. The notation is:

Branches Fields	Description
URL	The URL of the Branch
Name	The name of the Branch
`RepositoryName`	The name of the repository
`RepositoryURL`	The URL of the repository

#{for branches}

URL	${Branches[n].URL}
Name	${Branches[n].Name}
RepositoryName	${Branches[n].RepositoryName}
RepositoryURL	${Branches[n].RepositoryURL}

#{end}

Expand to see the sample code

```
#{for branches}
  ${Branches[n].URL}
  ${Branches[n].Name}
  ${Branches[n].RepositoryName}
  ${Branches[n].RepositoryURL}
#{end}

or

#{for <VariableName>=BranchesCount}
   Content and Issue Mappings. Example: ${Branches[VariableName].Field}
#{end}
```

Iterating Issue Pull Requests

As it is not known in advance how many pull requests exist for an issue, you can iterate a section over all the pull requests of an issue. This allows you to create a table that dynamically grows according to the number of existing pull requests. The notation is:

Pull Requests Fields	Description
URL	The URL of the Branch
Name	The name of the Branch
`RepositoryName`	The name of the repository
`RepositoryURL`	The URL of the repository
CommentsCount	Counts the number of comments in the pull request
Status	The status of the pull request
LastUpdated	The last time the pull request was updated
PullRequestReviewers.Name	The name of the pull request reviewer
PullRequestReviewers.Approved	Indicates the approval of the pull request reviewer

You can get information about reviewers from each pull request:

Pull Request reviewers Fields	Description
PullRequestReviewers.Name	The name of the pull request reviewer
PullRequestReviewers.Approved	Indicates the approval of the pull request reviewer

#{for pullRequests}

URL	${PullRequests[n].URL}
Name	${PullRequests[n].Name}
Id	${PullRequests[n].RepositoryName}
Author	${PullRequests[n].RepositoryURL}
CommentsCount	${PullRequests[n].CommentsCount}
Status	${PullRequests[n].Status }
LastUpdated	${PullRequests[n].LastUpdated}

Nested Iterations:

#{for m=PullRequests[n].PullRequestReviewers}

Name	${PullRequests[n].PullRequestReviewers[m].Name}
Approved	${PullRequests[n].PullRequestReviewers[m].Approved}

#{end}

#{end}

Expand to see the sample code

```
#{for pullRequests}
          ${PullRequests[n].URL}
          ${PullRequests[n].Name}
          ${PullRequests[n].RepositoryName}
          ${PullRequests[n].RepositoryURL}
        ${PullRequests[n].CommentsCount} (This represents the number of comments in a pull request)
        ${PullRequests[n].Status}
        ${PullRequests[n].LastUpdated}

        Here we have the PullRequestReviews where we can get all the reviewers for this pull request.
        #{for m=PullRequests[n].PullRequestReviewers}
                  ${PullRequests[n].PullRequestReviewers[m].Name}
                  ${PullRequests[n].PullRequestReviewers[m].Approved}
        #{end}
#{end}

or

#{for <VariableName>=PullRequestsCount}
   Content and Issue Mappings. Example: ${PullRequests[VariableName].Field}
#{end}
```

Iterating Issue Builds

Because it is not known in advance how many builds exist for an issue, you can iterate a section over all the builds of an issue. This allows you to create a table that dynamically grows according to the number of existing builds. The notation is:

Builds Fields	Description
ProjectName	The build project name

ProjectKey	The build project key

In order to get more information, you can get all the individual plans for the project and the corresponding build.

Build plans Fields	Description
`Plans.Key`	The plans build key
Plans.Name	Theplansbuild name
`Plans.BuildNumber`	The plans build number
Plans.BuildKey	The plans build key
Plans.BuildDuration	The duration of the build
Plans.BuildFinishedDate	The date when plans build was finished

#{for builds}

ProjectName	${Builds[n].ProjectName}
ProjectKey	${Builds[n].ProjectKey}

Nested Iterations:

#{for m=Builds[n].Plans}

Key	${Builds[n].Plans[m].Key}
Name	${Builds[n].Plans[m].Name}
BuildNumber	${Builds[n].Plans[m].BuildNumber}
BuildKey	${Builds[n].Plans[m].BuildKey}
BuildDuration	${Builds[n].Plans[m].BuildDuration}
BuildFinishedDate	${Builds[n].Plans[m].BuildFinishedDate}

#{end}

#{end}

Expand to see the sample code

```
#{for builds}
          ${Builds[n].ProjectName}
          ${Builds[n].ProjectKey}

        Here we have the each Build Plans where we can get all the individual plans for this project and the
correspondent build in existence for this plan.
        #{for m=Builds[n].Plans}
                  ${Builds[n].Plans[m].Key}
                  ${Builds[n].Plans[m].Name}
                  ${Builds[n].Plans[m].BuildNumber}
                  ${Builds[n].Plans[m].BuildKey}
                  ${Builds[n].Plans[m].BuildDuration}
                  ${Builds[n].Plans[m].BuildFinishedDate}
        #{end}
#{end}

or

#{for <VariableName>=BuildsCount}
   Content and Issue Mappings. Example: ${Builds[VariableName].Field}
#{end}
```

Iterating Issue Reviews

Because it is not known in advance how many reviews exists for an issue, you can iterate a section over all the pull requests of an issue. This allows you to create a table that dynamically grows according to the number of existing reviews. The notation is:

Reviews Fields	Description
Id	The review ID, e.g., 1
URL	The URL of the review
Status	The review status
Title	The title of the review
Author	The author of the review
Moderator	The moderator of the review

In order to get all the reviewers from this review, you can use the following mappings:

Reviewers Fields	Description
Reviewers.Username	The username of each reviewer
Reviewers.Completed	The completed name of each reviewer

#{for reviews}

Id	${Reviews[n].Id}
URL	${Reviews[n].URL}
Status	${Reviews[n].Status}
Title	${Reviews[n].Title}
Author	${Reviews[n].Author}
Moderator	${Reviews[n].Moderator}

Nested Iterations:

#{for m=Reviews[n].Reviewers}

Name	${Reviews[n].Reviewers[m].Username}
Completed	${Reviews[n].Reviewers[m].Completed}

#{end}

#{end}

Expand to see the sample code

```
#{for reviews}
          ${Reviews[n].Id}
          ${Reviews[n].URL}
          ${Reviews[n].Status}
          ${Reviews[n].Title}
          ${Reviews[n].Author}
          ${Reviews[n].Moderator}

        Here we have the Reviewers for each review where we can get all the individual reviewers for this
review.
        #{for m=Reviews[n].Reviewers}
                 ${Reviews[n].Reviewers[m].Username}
                  ${Reviews[n].Reviewers[m].Completed}
        #{end}
#{end}

or

#{for <VariableName>=ReviewsCount}
   Content and Issue Mappings. Example: ${Reviews[VariableName].Field}
#{end}
```

Applying filters to Iterations

If you want to take the previous iterations over comments, subtasks and issue links to another level of control, you can use a JavaScript filter to define over which issues the iteration will be made. This can be useful in the following scenarios:

- Iterating over linked issues that are only of a specific issue type
- Iterating over subtasks of a specific issue type
- Iterating over linked issues with a specific priority
- Iterating over comments created by a specific user

The notation for applying filters to the iterations is:

Expand to see the sample code

```
#{for <VariableName>=<LinksCount|SubtasksCount|CommentsCount|WorklogsCount>|filter=%{<Javascript>}}
   Content here
#{end}
```

- **VariableName** is the name of the variable to use as the iteration index.
- **LinksCount|SubtasksCount|CommentsCount** indicates over which type of entities you want to iterate.
- **Filter** indicates the filter to be applied in the iteration.

Notice that as the filter is evaluated as a JavaScript expression, which provides flexibility in the definition of the conditions. You can use and (&&), or (||) and other logical operators supported by the JavaScript language.

It is also possible to format fields inside iteration filters. For more information on formatters, see Iterations.

The image below demonstrates an example of a template that iterates over issue links and comments with filters being applied.

Links Bugs with High Priority:

#{for n=LinksCount|filter=%{'${Links[n].IssueTypeName}'.equals('Bug') &&
'${Links[n].Priority}'.equals('High')}}

Key	${Links[n].Key}
Summary	${Links[n].Summary}
Link Type	${Links[n].LinkType}

#{end}

Nested Iterations:

#{for j=LinksCount}

Comments for Linked Issue ${Links[j].Key}

#{for m=Links[j].CommentsCount}

Author	${Links[j].Comments[m].Author}
Body	${Links[j].Comments[m].Body}
Date	${Links[j].Comments[m].CreatedDate}

#{end}

End of Comments of Linked Issue ${Links[j].Key}

#{end}

For a working example of this functionality, check the template Sample Iterations in the Template Store.

Iterating in the same line of the document

You can also possible to iterate values in the same line of the document. This can be useful if you want to display a list of Subtasks on Linked Issues in the same line, separated by commas or spaces.

Expand to see the sample code

```
Users that added comments to this issue: #{for comments}${Comments[n].Author} #{end}

Subtasks of this issue: #{for j=SubtasksCount}${Subtasks[j].Key};#{end}

Linked issues this issue duplicates: #{for j=LinksCount|filter=%{'${Links[j].LinkType}'.equals
('duplicates')}}${Links[j].Key} #{end}
```

Iterating in the same cell in an Excel document

You can also iterate values in the same cell in an Excel document. You can achieve this by simply making your Iteration inside the same cell.

You can use all the Iterations that you are used to and construct them in the exact same way, the difference being that you only use one cell to do them.

Expand to see the sample code

```
Issue iteration as a demonstration.
Copy this iteration below and paste it into a cell.

&{for issues} ${Key} &{end}
```

Iterating with the BREAK or CONTINUE statement

You can iterate anything, set up a Conditional expression and then utilize the BREAK and CONTINUE statements.

The way to do this is by doing a normal Conditional expression and using the mapping #{break} or #{continue} inside it.

Expand to see the sample code

```
Imagine that you have a Jira Issue that contains these comments:
- Hello
- World
- Greetings
- Hi

For the Break functionality, lets say that you want to stop the iteration if the current comment is "World".
Here is the template for that:
#{for comments}
Current Comment: ${Comments[n].Body}
#{if (%{'${Comments[n].Body}'.equals('World')})}
#{break}
#{end}
Current Comment Author: ${Comments[n].Author}
#{end}
In this case, Xporter for Jira will print the comment "Hello" and it´s author. Next it will print the comment
Body "World" but since the Conditional expression is true, it will stop the iteration all together and not
print anything else.
Note: Anything after the #{break} mapping will not be printed in the exported document.

For the Continue functionality, lets say that you want to skip to the next iteration if the current comment is
"World", bypassing the Author mapping for this iteration. Here is the template for that:
#{for comments}
Current Comment: ${Comments[n].Body}
#{if (%{'${Comments[n].Body}'.equals('World')})}
#{continue}
#{end}
Current Comment Author: ${Comments[n].Author}
#{end}
In this case, Xporter for Jira will print the comment "Hello" and it´s author. Next, it will print the comment
Body "World" but since the Conditional expression is true, it will continue to the next iteration, not printing
the Author of the "World" comment.
```

Iterating Parent Issues

You can iterate a section over all the parent issues of an issue. This allows you to create a table that dynamically grows according to the information you want to see from parent issues.

Imagine that you have a Jira Issue that contains a Key, Summary, Description and further information. From now on, you are able to get all the information from a parent issue. In order to get those fields, you just need to have the following definition:

`${Parent.<Field>}`

Example:

Expand to see the sample code

```
&{for issues|filter=%{'${IssueTypeName}'.equals('Sub-task')}}
   ${Parent.Key}
   ${Parent.Summary}
   ${Parent.Description}
   ${wiki:Parent.Description}
   ${html:Parent.Description}
   ${dateformat(“dd-MM-yyyy HH:mm:ss”):Parent.date}
   ${emailaddress:Parent.userpicker}
&{end}

This example only has a few fields, but this new feature allows you to get all information from a parent issue.
```

Sorting iterations

Imagine that you have an iteration and want to sort it by any field that it can export normally. This will be the header for such an iteration:

```
#{for comments|sortby=<Iteration mapping>}
```

NOTE: The mapping after the "sortby" must be equal to the supported mappings for each Iteration.

Example:

Expand to see the sample code

```
This iteration will be sorted by the Body of all the comments in the issue.

#{for comments|sortby=Body}
${Comments[n].Author}
${Comments[n].Body}
#{end}
```

Sort By Bulk export

The sortby can also be used to sort a &{for issues} iteration on a Bulk Export.

Expand to see the sample code

```
&{for issues|sortby=IssueTypeName}
${Key} - ${IssueTypeName}
&{end}
```

Sorting Criteria

asc and **desc** can be defined in order to define how do you want to sort your data. The default value is **asc**.