Oracle® Documaker

Regression Impact Guide Version 13.0.2

G40412-02
October 2025

Oracle Documaker Regression Impact Guide Version 13.0.2,

G40412-02

Contents

1 Overview

Oracle Documaker is a Customer Communication Management (CCM) suite of products that enables organizations to dynamically create, manage, publish, and deliver adaptive enterprise content throughout the business life cycle across all locations and lines of business. It offers a cost-effective way to address the design, production, and multichannel delivery of a broad spectrum of documents from highly structured transactional documents delivered in high-volume batch to highly personalized interactive correspondence delivered on-demand. Oracle Documaker's rule-driven process transforms data into personalized and precise mission-critical enterprise customer communications. Oracle Documaker helps businesses improve customer service, get to market faster, reduce risk of improper communication, maximize effective customer contact, and reduce costs.

Visit us online at http://www.oracle.com/us/products/applications/documaker/overview/index.html for more information on how Documaker can improve your business operations.

How this guide can help you?

This guide provides you high-level information about Oracle Documaker updates and changes when upgrading from a prior Oracle Documaker version 13.0.0 to Oracle Documaker 13.0.2.

What's unique about this guide?

This guide assists you in planning your upgrade from Documaker Release 13.0.0 to Oracle Documaker Release 13.0.2. Each chapter in the guide includes details on changes that affect generated output, user interface, etc.

How is this guide structured?

This guide structured in such a way that you may find quick answers to your questions during upgrade. Following this document you will notice each issue and feature contains 4 types of questions:

What changed?

This section describes the change at high level.

Why did it change?

This section describes if the change is result of a bug fix or a new feature introduced in the current release.

How does this change may affect you?

This section describes if the change is a bug fix, you may need to modify few settings. For e.g. INI setting or add\remove ( ) in a DAL script, etc.

Are changes needed to your setup for this change to take effect?

This section describes the setup that illustrates the change.

General Issues

The enhancements in 13.0.2 all require some enablement by utilizing new options.

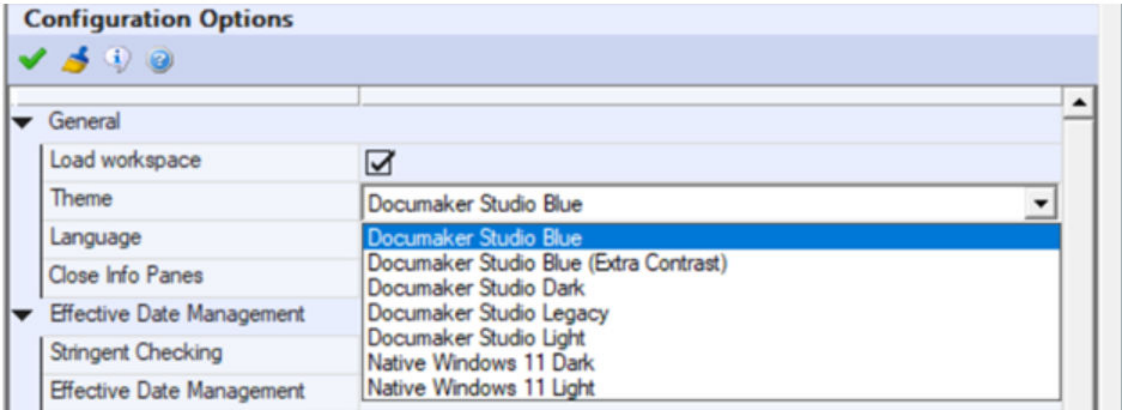
The features in this release are not expected to impact regression tests or existig behavior.

To enable and leverage the new features in 13.0.2 please refer to the Release Notes.

2
Documaker Studio Theme Refresh

New UI themes have been added to Documaker Studio offering blue, light, and dark display options and older (outdated) themes have been eliminated.

Themes are available for change under the View/Options/Configuration Options.



The prior **View** menu options for Color Scheme and Context Color changes have been eliminated as these attributes are now controlled by the underlying theme choice.

The new default theme is the Documaker Studio Blue theme, but individuals may wish to review the others to determine which they prefer. The prior default (blue gradient) theme is available as the Documaker Studio Legacy theme.

The UI refresh includes new icon/bitmaps on the toolbars and ribbons offering better resolution and clarity.

DAL Updates

There have been various updates to DAL functionality that are deemed corrections or improvements.

- **DelSubform**
  This function is meant to locate and remove a named subform. Previously, if the script named a subform that could not be located the entire transaction and even the application might terminate. This has been corrected. If a call to DelSubform does not locate the requested item, the function returns the 0 result.
- **SpanField**
  This function moves a field between two other fields (the left and right fields) to “span” the distance with a fill character.

  If called when the field on the left was empty (no data), the span field function would incorrectly assign the fill character to the wrong field. This has been corrected.

  If both the left and right field are empty (no data), the span field will also be empty – effectively suppressing the span.

- **PatchRelease**
  This function previously returned an incorrect value as the patch number. The Documaker version number takes this form.

  Full Version Number = A.B.C.D.E (e.g., 13.0.2.0.37200):

  – A is Major Version number of the release (e.g., 13 in example above)
  – B is Minor Version number of the release (e.g., 0 in example above)
  – C is Minor Release number of the release (e.g., 2 in example above)
  – D is Patch Release number (e.g., 0 in example above)
  – E is Build Release number (e.g., 37200 in example above)

  The PatchRelease function should return the fourth component of the full version number.
- **FieldRule**
  Omitting the field length will now default to use the defined field length on the section. This change should help in those cases where the ultimate field length may not be known by the script writer and prevent script changes if the field length is later changed in the section.
- **StrCompare**
  The StrCompare should return values 0, -1, or 1 depending upon whether the strings are equivalent or one is lexicographically “lessor” or “greater” than the other. Previously, executing the same script on Windows versus Linux could return other values. The platform differences have been eliminated.
- **Cut**
  With the release of 13.0.x (64-bit), the Cut function interpreted the Position parameter of 0 incorrectly. This change restores the behavior of position 0 to work as before.
- **DBUnloadDFD**
  On Linux platform this function could fail because of improper handling if subfolder names were included. This has been corrected so that Windows and Linux should manage the path name appropriately.
- **PathExist**
  On Linux platform this function could fail because of improper handling if subfolder names were included. This has been corrected so that Windows and Linux should manage the path name appropriately.

Documaker Workstation (AFEMain) AutoSave Timer Recovery Problem

Documaker Workstation is also known by other names: Documaker Desktop; AFEMain; or Policy Production System (PPS). This discussion will simply refer to the application as AFEMain.

AFEMain environments uniquely identifies WIP transactions. This unique identifier is kept in the WIP table and matches the name on files commonly referred to as the NA and POL data files stored in the WIP directory.

The standard identifier utilized by AFEMain on WIP transactions is a GUID comprised of a mix of letters and digits – generally 32 characters in length. This ID method has been in place for many years.

Prior to the use of GUIDs, the AFEMain WIP transactions were uniquely identified by increasing a simple number represented using 8 characters.

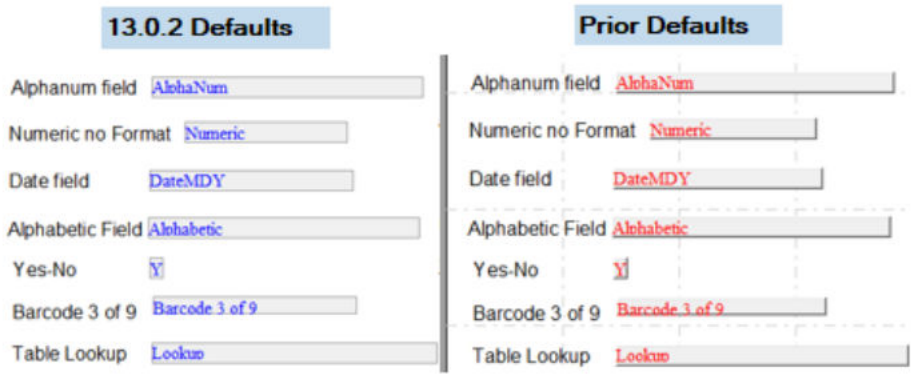
Documaker continues to support the legacy unique numbering method if the customer WIP table definition has not been upgraded.

As of this version 13.0.2, it was discovered that the optional Auto-Save functionality only worked with the legacy WIP transaction numbering method. This has been corrected.

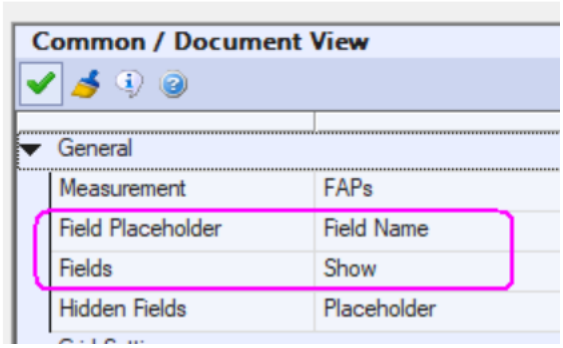
Customers that are using AFEMain to manage WIP transactions and using Auto-Save do not have to make any changes to inherit this correction after upgrading. An auto-saved transaction should restore a prior WIP session that terminated improperly whether your setup uses the standard GUID naming method or continues to use the legacy unique numbering method.

Field Templating Options

Field representations when editing and viewing forms and sections now uses a “flat” shaded area and utilizes blue text. The prior default color of fields (not specifying a color) was red and shown in a raised “button-style” depiction.

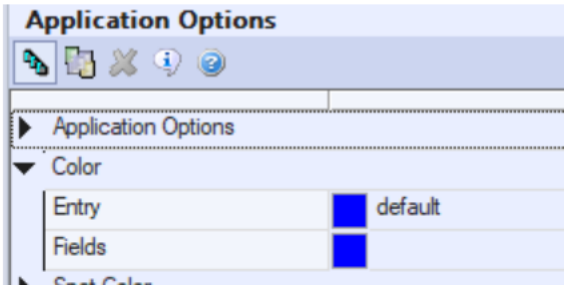


Options available to change the representation of fields within Studio were in different areas of the configuration. New in 13.0.2, these presentation options have been brought together under the View/Options/Common-Document View options.



Combinations of these two options offers presentation methods that someone might prefer over the default.

The default field color – for fields that do not otherwise have a color assigned – is an option that can be specified for the entire workspace within the Application (BDF) Definition. This has not changed in 13.0.2 other than the default will now be blue if another color was not already chosen.



Handling of Multi-page Section Occurrence During Field Mapping

Previously, if a section became a multi-page section due to overflow of a text area, each page of that section would have been seen as a new occurrence within the form. This made it infinitely difficult to map fields embedded within the flowing text as the mapping rule needed to know the section occurrence. Now, these overflowing/multi-page sections are treated as the same occurrence.

For example: If you have one section that overflows to three pages, the three-pages of that section will now all be treated as "occurrence=1". If another section with the same name appears later, it will be considered the second occurrence.

In this way, xpath statements that utilize occurrence indicators will find the proper section when field mapping of data occurs.

Here is a more detailed example.

Assume multiple occurrences of a section are triggered to match the number of matching Coverage nodes found in the input XML.

```
<Data>
<Coverage>Coverage is prolonged beyond its original term, such as for a
travel insurance policy when a return date is delayed.
</Coverage>
<Coverage> Protection is added for specific risks not typically included in a
standard policy, such as from explosions, hail, or vehicle damage.
```

```
</Coverage>
</Data>
```

It would not be unusual for this type of data to be mapped into a multiline text (MLT) field on this type of section. This field will grow to accommodate the text and might split to create a new page if the position of the section is near the bottom of a page and grows too large.

The Xpath mapping of this field might look like this: `\Data\Coverage[****]`

The four asterisks [****] is a placeholder where an occurrence value will be substituted. The actual occurrence used will depending upon whether you have indicated this to use the form set, form, or section occurrence within the form.

Because there are two sections, you might expect the two mapping to resolve like this:

```
\Data\Coverage[1] – on the first occurrence
\Data\Coverage[2] – on the second occurrence
```

This is now the expected behavior whether the first section overflowed to a subsequent page or not.

In prior versions, the second section would have remained empty if the first section overflowed the page. This was because the counting occurred like this:

```
[Coverage Section with MLT field]
Coverage is prolonged beyond its original term, such as for a travel
insurance
<new page>

[Coverage Section with MLT field] [cont]
policy when a return date is delayed.
[Coverage Section with MLT field]
Protection is added for specific risks not typically included in a standard
policy, such as from explosions, hail, or vehicle damage.
```

When the original section overflows to a new page, another copy of that section is inserted to allow the continuation/overflow. Depending upon the amount of data, it is possible that a single MLT field might span any number of pages.

By strictly counting the section occurrence then, the prior versions would have seen this as three occurrences of the section [Coverage Section with MLT field]. That means the second mapping of data would have incorrectly used the Xpath: `\Data\Coverage[3]`, which would have resulted in no data being returned.

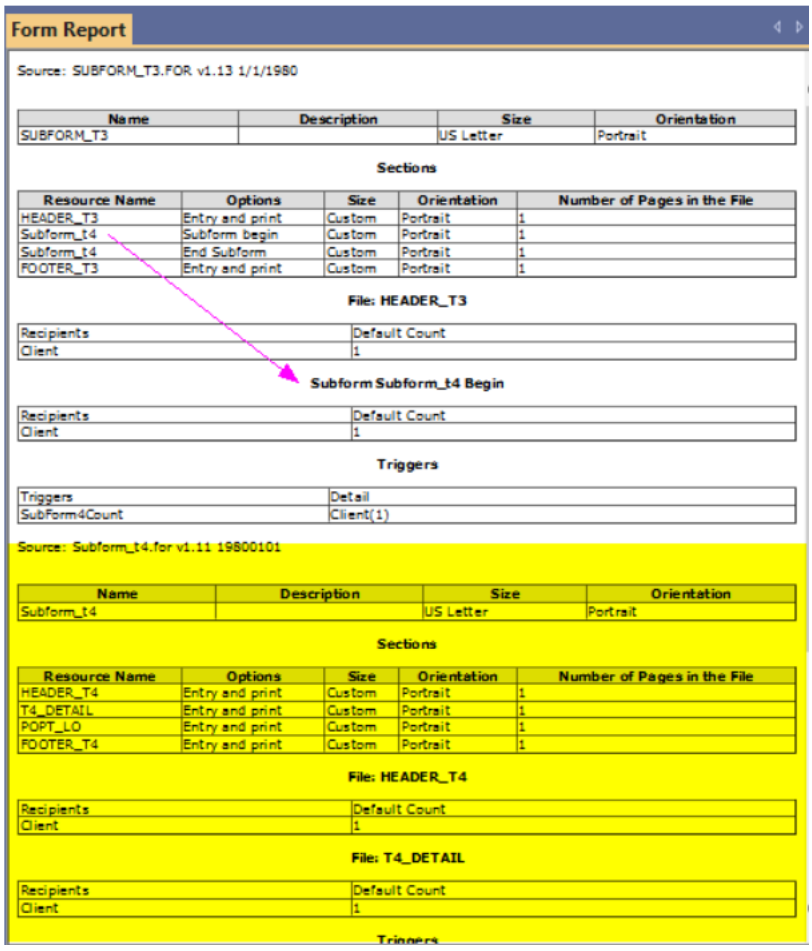
It is for this reason that the overflow/occurrence mapping of multi-page sections has been changed to look for the independent sections that triggered and not rely upon a strict count of the sections. This allows the functionality to use the correct Xpath `\Data\Coverage[2]` when mapping data occurrences.

3

DMStudio Form report Now Including Details of Subform

If a form contains a subform, previously the Form report only listed the subform and not the details within that subform. With this update, the subform details which includes additional sections and triggers which will be included in the report. This functionality is recursive and will resolve subforms found within child subforms.

The new details will follow the section that defines the beginning of the subform.



When the subform details end, the report continues with the section details of the original form.

4

HTML Import of Lists Now Indents a Subordinate List

Prior to this version, if a list contained a subordinate list the result did not indent the child list. Consider this example:

```
<p><b>Fruits</b></p>

<ol>

<li>Apple</li>

<li>Banana</li>

<li>Citrus</li>

<ol type='a'>

<li>Grapefruit</li>

<li>Lemon</li>

<li>Orange</li>

</o>

<li>Pear</li>

<li>Strawberry</li>

</ol>
```

Keep in mind that the source view above has been indented to aide understanding and does not control the final presentation.

The current result of importing this markup will look like this:

Fruit

```
1. Apple
2. Banana
3. Citrus
   a. Grapefruit
   b. Lemon
   c. Orange
4. Pear
5. Strawberry
```

Where in prior versions the result would have looked like this:
`eibccbjgtjkegcnkvhddujnckidijufjeeehgelniufr`

Fruit

```
6. Apple
7. Banana
8. Citrus
a. Grapefruit
b. Lemon
c. Orange
4. Pear
5. Strawberry
```

Any existing implementation using subordinate lists within another will now generate indented output.

5

HTML Import of Ordered List

Ordered list default to a number bullet. The type of bullet for an ordered list is controlled by the “type” attribute, where `<ol type= character>`

```
<ol type=1> numbers (e.g., 1. 2. 3. – 22. 23. 24. )

<ol type=a> lowercase letters (e.g., a. b. c. – x. y. z. )

<ol type=A> uppercase letters (e.g., A. B. C. – X. Y. Z. )

<ol type=i> lowercase Roman numerals (e.g., i. ii. iii. – xiii. xiv. xv. )

<ol type=I> uppercase Roman numerals (e.g., I. II. III. – XIII. XIV. XV. )
```

An ordered list will default to begin at the first “number” for the type of bullet. So a numbered list will start at 1. A letter list will start with the letter a or A. Roman numeral will start with i or I. The starting value can be changed by specifying a start= attribute. The start attribute is always a number, even if the resulting bullet is not literally a number. So if you want to start a lettered bullet list at the letter D, you would specify as:

```
<ol type=A start=4> uppercase letters starting at 4 (e.g., D. E. F)

<ol type=i start=6> lowercase Roman numbers starting at 6 (e.g., vi. vii.
viii. )
```

6

HTML Import of Ordered List Supports a Start Value

An ordered list defined to use numbers or letters will also support the start= attribute.

Example

```
<p><b>Fruits</b></p>

<ol start= "4">

<li>Apple</li>

<li>Banana</li>

<li>Lemon</li>

<li>Pear</li>

<li>Strawberry</li>

</ol>
```

Will result in this output:

Fruit

```
4. Apple
5. Banana
5. Citrus
6. Pear
7. Strawberry
```

Any legacy implementation ignored the start= attributes, but now recognized in this version.

Keep in mind that in HTML, the start= value is always a number. This applies even if your list type is specified to use letters or Roman numerals. Here is the same example, but with letter bullets.

```
<p><b>Fruits</b></p>

<ol type="A" start= "4">

<li>Apple</li>

<li>Banana</li>

<li>Lemon</li>
```

```
<li>Pear</li>

<li>Strawberry</li>

</ol>
```

Will result in this output:

Fruit

```
D. Apple
E. Banana
F. Citrus
G. Pear
H. Strawberry
```

7

HTML Import of Unordered List

An unordered list default to a filled-circle (disk) bullet. The type of bullet for an ordered list is controlled by the “type” attribute. Documaker continues to supports a legacy type definition that follows:

```
<ul type=″C″> circle (open)

<ul type=″S″> square (solid)

<ul type=″D″> disc (filled circle)
```

Starting in this version, the HTML import also supports the standard “list-style-type” attribute definitions. Use of these definitions in an existing input file were previously ignored and will now be honored.

```
<ul type=″list-style-type:circle″> circle (open)

<ul type=″list-style-type:square″> square (solid)

<ul type=″list-style-type:disc″> disc (filled circle)
```

8

HTML Print with Color Text

Testing revealed that the HTML print output incorrectly included color on some text output that did not define the “Print in Color” attribute. Color shown during the design is only to be used in print when that attribute is set.

If some HTML output is no longer showing color for some text elements, the original section (FAP) will need to be edited for possible correction.

9

Improved Mapping HTML Markup from XML into Multi-line Text Fields

There has long been some support for mapping MLT fields using (limited) HTML markup from XML. The simplest situation can designate paragraphs using the <p> tag and bulleted lists using the ordered-list <ol> or unordered-list <ul> with list-item <li> tags. Also supported were text modifiers like bold <b>, italic <i>, and underline <u>.

Enhancements have been added to the markup support and some of these changes may impact your prior regression results.

- Recognition of paragraph tags is case insensitive.

The original implementation of this functionality years ago defined most tags in uppercase and checking for lowercase equivalent tags was inconsistent. For instance, a paragraph tag was expected to be <P> rather than <p> as is the present standard. During this review, tag identification has been validated to be case insensitive. This means that a legacy implementation may now result in different content if there were occurrences of lowercase tags that were previously being ignored. Recognition of these previously ignored paragraph tags could result in your content generating additional lines of spacing although the text content will likely remain consistent.

- The content of a MLT field should default to the font and color specified on the field.

Unless overridden by a specific attribute in the markup, the font and color of the text should correspond to that specified by the field. This was not happening in prior versions but will going forward. Alternately, you can start including font or color modifier tags in the markup.

- The Font <font> tag is considered obsolete but still supported.

The legacy support for the font <font> tag with attributes “face=” and “size=” are still supported, even though the font tag is considered obsolete starting in HTML5. However, one significant difference starting this version is that the “size=” modifier no longer assumes the number following is a point size.

For example, in prior versions a tag <font size=”2”;> would have assumed you wanted a very small 2-pt font size. This is now more correctly identified as indicating you want a “step down” from the default font size which is assumed to be a 12-pt font. HTML specification defined the font size= attribute as a range from 1 – 7 where 3 was assume the default 12-point font size. This table indicates the size assumption.

Size=”n”	Point Size
1	8
2	10
3	12
4	14
5	16
6	20
7	24

The actual font size selection in your setup will depend upon the font cross-reference (FXR) file. For instance, if your current font family does not have a 10-point size defined in the FXR, the system may choose a size larger or smaller.

As noted, the font tag will still be recognized in the markup, but consider having new mappings use the supported tag rather than <font>.

- Span `<span>` tag is now supported.

New to this version is support for the tag. A span can be used to define attributes to change the font style include the font-family and the font-size. If your markup previously included tags, these were being ignored, but will now alter your output.

This is the preferred replacement for the <font> tag that is now obsolete starting with HTML5. Note that legacy support for the <font> tag will continue and has not been removed. For more information refer to the font tag section of this document.

Description of specific attributes supported in the `<span>` tag are listed separately in this document.

- Font-Style attribute supported for all paragraph-level tags (`<p> <br> <li>`) and `<span>` tag.

In this version, the style= attribute modifier for paragraph or span tags will all parse and interpret the font-family, font-size, and color attributes in the same way. In addition, the modification of these attributes will be limited to the “scope” of the tag to which these are associated. This means that at the closing tag, the prior text styling attributes that were in existence before this tag began will be restored.

Example

```
<p style="font-family: Arial; font-size: 14pt; color: #0000FF;">This text
should be Arial 14pt and blue</p>
```

This paragraph tag specifies a font change for both the name of the font and the size. In addition, the text color has also been changed.

The style attribute is a quoted string that specifies one or more modifiers separated with a semicolon. The attributes may be specified in any order or omitted as necessary.

Attributes not included are assumed to be “inherited” from what was established before this tag began. So a tag of `<span style=″color: green;″>` will change the text color to green, but does not modify whatever font name or size had previously been defined. At the close of this span `</span>` the color will revert to whatever color had been established before this span began.

Please note that this behavior of restoring the prior attributes at the close of the associated tag might yield a regression difference to legacy behavior.

- Color attribute now supports color name keywords for all paragraph-level tags (`<p> <br> <li>`) and `<span>` tag.

Legacy markup mappings that recognized color only supported the hexadecimal `#rrggbb` format. New to this version, support for standard html color names has been added. In general, your use will probably be for standard color names like red, green, blue, black, cyan, yellow, magenta, etc. The full list of supported names is extensive. An example list can be found here: https://www.w3schools.com/colors/colors_hex.asp

- Font-size attribute now supports measurements other than points (pt) supported for all paragraph-level tags (`<p> <br> <li>`) and `<span>` tag.

Legacy interpretation of the font-size attribute assumed the reference was a point size (pt) even if the value specified an alternate measurement descriptor. In addition to point (pt), this version introduces support for font-size specified in centimeter (cm), millimeter (mm), inches (in), pixels (px), picas (pi) and em (from em quadrat). This discussion will not go into a detailed explanation, but please recognize that the sizes specified with inches or centimeters likely require decimal values like 0.12in or 0.5cm.

This new support for the measurements might be revealed in regression testing. For instance, if the markup had specified “font-size: 16px”, a prior version would have interpreted this as requesting a 16-point font size. With this new support, this request would result in a 12-point font selection as this is what 16 pixels indicates.

- Font-size attribute now supports keyword size description for all paragraph-level tags (`<p> <br> <li>`) and `<span>` tag.

Keyword	Point Size
xx-small	4
x-small	6
small	10
medium	12
large	14
x-large	18
xx-large	24

The actual font size selection in your setup will depend upon the font cross-reference (FXR) file. For instance, if your current font family does not have a 6-point size defined in the FXR, the system may choose another adjoining size – smaller or larger.

- Support for HTML character entities has been added.

Character entities are escaped using the ampersand & followed by a keyword, decimal value, or hexadecimal value. Standard ASCII characters fall in the range from 32-255. Documaker support for Unicode characters range up to decimal 65535 or hexadecimal 0xFFFF.

Any ASCII character may be represented in HTML using a decimal or hexadecimal entity. For instance the capital letter A could be represented as `&#65;` or `&#x41;`. There is generally no reason to use a character entity for most standard text. However, because HTML (and XML) is often encoded as UTF8, certain ASCII characters greater than code-point 127 must be represented with a character entity. In addition, certain characters are interpreted to delineate structural components of HTML/XML. For instance, less than < and greater than > symbols are used to delineate nodes in the content. The quote is used in XML and HTML to delineate attributes or other components. If these “control” characters are instead meant to be part of the text element, these characters need to be escaped using a character entity.

ⓘ Note

- Documaker does not support all possible HTML character entity names or representations. Only a subset of entities are supported. These will be described here.
- The HTML markup is provided via XML extract and certain HTML escapements may be misinterpreted by the XML parser. To ensure your markup is bypassed by the XML parser, you should escape your escapement.

> **Note**
>
> & < and > are all recognized by XML as character entities. If the text is meant to only apply to the HTML markup, your references should be depicted as: & < > in the XML.
>
> This first & will be recognized and converted by the XML parser to ampersand without trying to interpret the entity meant for the HTML markup. When the content is later mapped to a MLT field, the HTML markup text references will be represented as expected.

As noted earlier, any standard ASCII character can be represented as a decimal or hexadecimal value. Decimal representations start with the hash #, followed with up to 3-digits for standard ASCII codepoints or 5-digits for Unicode codepoints. Hexadecimal values also start with hash but followed by the small-letter x and then the hex representation for that codepoint.

Examples:

The copyright symbol © can be represented as `&#xa9;` or `&#169;`

The greater than symbol > can be represented as `&#x3e;` or `&#62;`

There are keyword designations for common character entities that may be easier to use and recognize.

Character	Name	Decimal	Hex	Comment
"	&quot;	&#34;	&#x22	Double quote
&	&amp;	&#38;	&#x26	Ampersand
‘	&apos;	&#39;	&#x27	Apostrophe
<	&lt;	&#60;	&#x3c	Less than
>	&gt;	&#62;	&#x3e	Greater than
				Hard space (non-breaking space)
¢	&cent;	&#162;	&#xa2	Cent
£	&pound;	&#163;	&#xa3	Pound
¤	&curren;	&#164;	&#xa4	Currency
¥	&yen;	&#165;	&#xa5	Yen
¦	&brvbar;	&#166;	&#xa6	Break vertical bar
©	&copy;	&#169;	&#xa9	Copyright
®	&reg;	&#174;	&#xae	Registered trademark
°	&deg;	&#176;	&#xb0	Degree
±	&plusmn;	&#177;	&#xb1	Plus-Minus
²	&sup2;	&#178;	&#xb2	Superscript 2
³	&sup3;	&#179;	&#xb3	Superscript 3
´	&acute;	&#180;	&#xb4	Acute
¶	&para;	&#182;	&#xb6	Paragraph mark
·	&middot;	&#183;	&#xb7	Middle dot
¹	&sup1;	&#185;	&#xb9	Superscript 1

Character	Name	Decimal	Hex	Comment
¿	¿	¿	&#xbf	Inverted question mark
÷	÷	÷	&#xf7	Divide

Any entity not recognized not recognized as a keyword or is formatted improperly will be left undisturbed in the content. If you see a representation that did not convert to the character you expected, be sure to check the formatting and/or spelling of the element.

10

ODEE PDF File Sizes for Scheduled Batch

ODEE PDF output from scheduled batches may have larger file sizes than necessary. This has been corrected. No user changes are required, and there will be no changes in PDF visual appearance.

11

Relax DAL Date Validation During Rule Mapping

An argument can be made that assigning data that does not meet the field's definition should be an error. During manual entry, this validation is indeed enforced. However, many rules that map data do little in the way of validation before assigning to the field. This legacy behavior continues such that we do not invalidate customer resources that have been long accepted. For instance, a move_it rule can map text data into a field declared numeric and likewise any text into a field declared as a date. For many legacy rules, this is not considered an error during field mapping, even though the data may be contrary to the field's definition. Succeeding rule additions, like AnyToAny, does utilize the field's formatting attributes and does validation.

The DAL SETFLD() function also relies upon the calling application (entry versus rules) to determine whether to accept data assigned by DAL script. The recent feature providing date field future/historic attributes added date validation for all DAL type fields. This is true whether the date field used the new attributes. This means a legacy environment may encounter a DAL error if a DAL script attempts to assign non-date value into a date field using the SetFld() function.

If you encounter this date validation error from rule mapping a date field using a script, you have two options.

- Change the field type within the affected section to be alphanumeric instead of a date.
- Add a new INI option to relax the validation and preserve the legacy behavior.

If you change the field to alphanumeric, it can accept any data. Committing the resource with the field change means the problem goes away without having to fix the script.

If you don't wish to change the resource field declaration, a new INI option is available to relax the validation. Remember this option only affects the execution of DAL scripts called from Gendata/Genprint. As mentioned earlier, manual data entry would have always enforced date validation, whether assigned from a script or not.

```
<Gendata>
RelaxDALDateValidation = Yes
```

The default is not to relax validation because it is assumed a field's type should validate the data. Adding this option set to "Yes" will relax the validation to what was previously allowed. Remember it only applies during Gendata/Genprint processing.

12

Set Language for Specific Text in PDF

Specifying the language for text within a PDF will improves accessibility, especially when the document contains many languages. Setting language attributes improves the experience for users relying on screen readers, allowing them to interpret content accurately. When the reader encounters a change in language for a given text element, the spoken tone and inflection should change to make the content more recognizable to the user.

No noticeable difference should occur in the actual printed document (PDF), but some of the intermediate files utilized by the system will reflect the new language attribute.

Inlined text records in NA file will have one additional element for accessibility language ID. Although the new attribute will only appear when the section has been changed to include it, any regression of legacy transactions will contain the placeholder for this attribute.

Old

```
TP417BX1;8334;3806;19134;3866;912;0;0;0;0;0;0;0;0;;1;0;\OOEDGEKLC...
```

New

```
TP417BX1;8334;3806;19134;3866;912;0;0;0;0;0;0;0;0;;1;0;;\OOEDGEKLC...
```

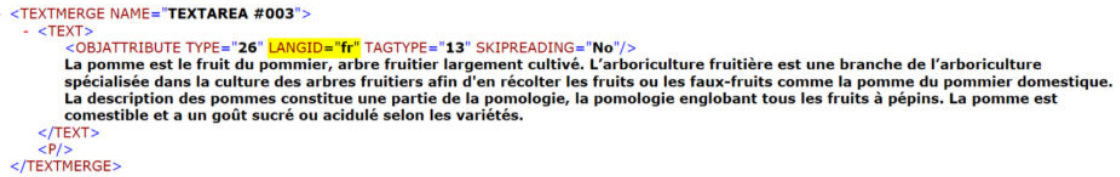
Accessibility attribute records (A,xA26) in sections (.FAP files) for text labels, variable fields, text areas, etc. will have one additional element for an accessibility language ID.

For example,

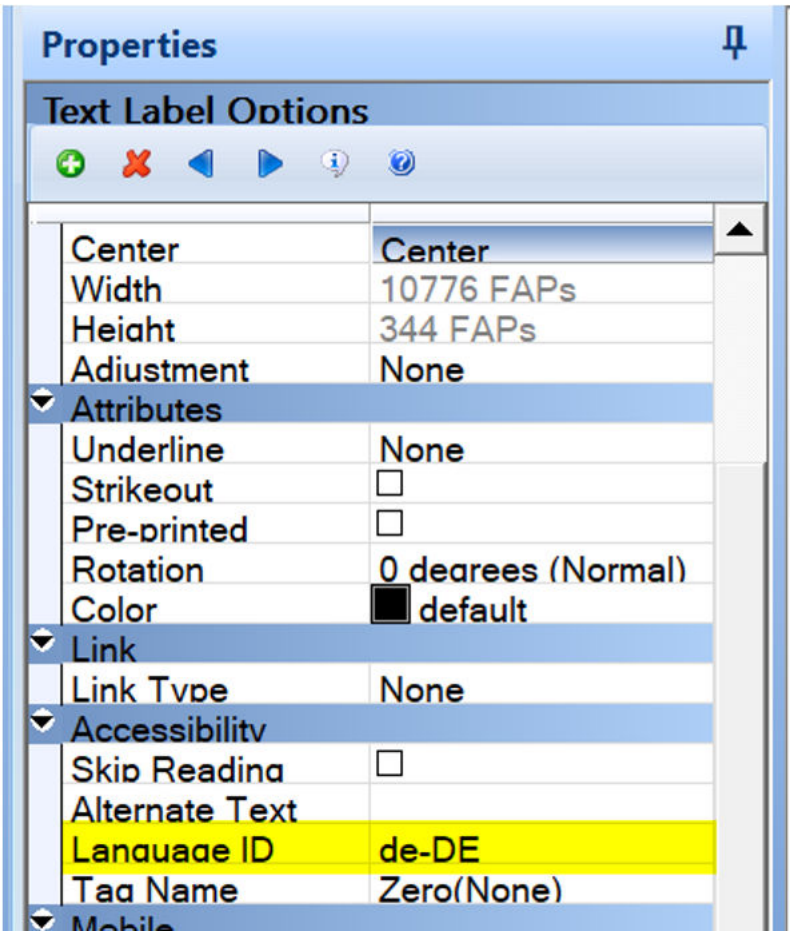
- A,TA26,0,13,"fr"
- A,FA26,0,0,"en-US"
- A,MA26,0,0,"es"

Baseline regressions may need to be updated to reflect the additional attribute position.

Accessibility attribute records in XML output for text labels, variable fields, text areas, etc. will have a `LANGID` element for an accessibility language ID.



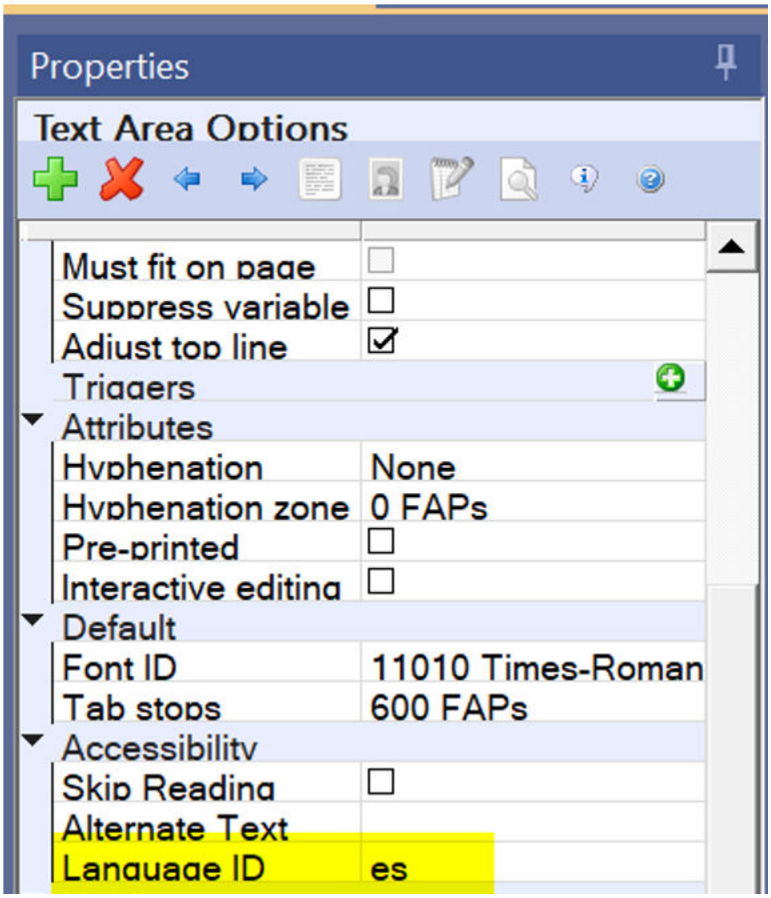
Setting the Language ID in the Accessibility fields in Studio



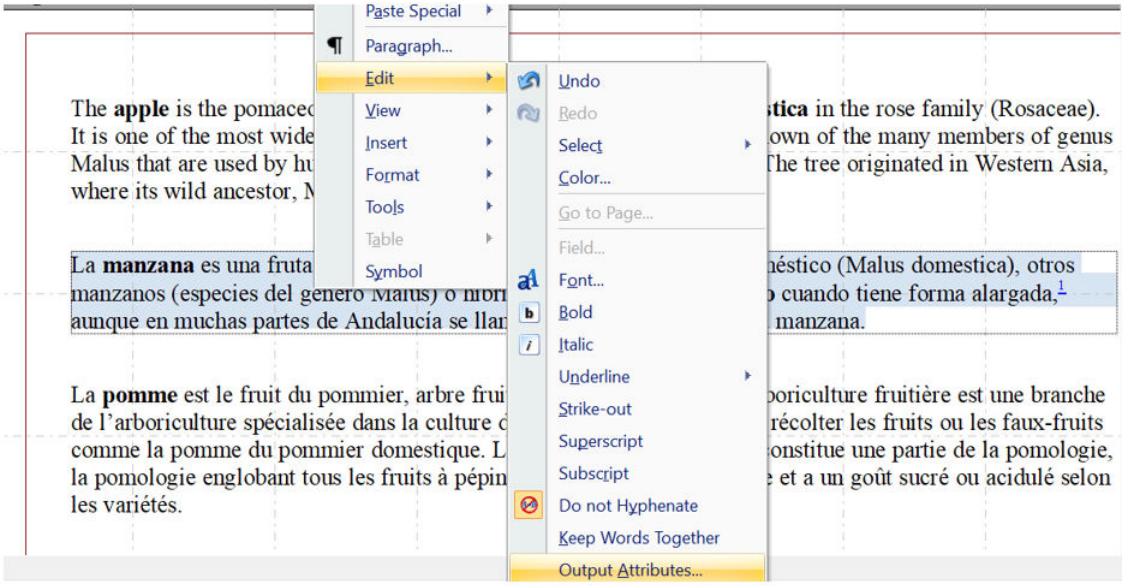
In the example above, the text label that contains German text is set with a Language ID of de-DE, representing German as spoken in Germany. The `de` represents the ISO 629 language code for German and the `DE` represents the ISO-3166 Country Code for Germany.

It is not required to use the country code, user can use only `de` as the ISO 629 language code for the German language.

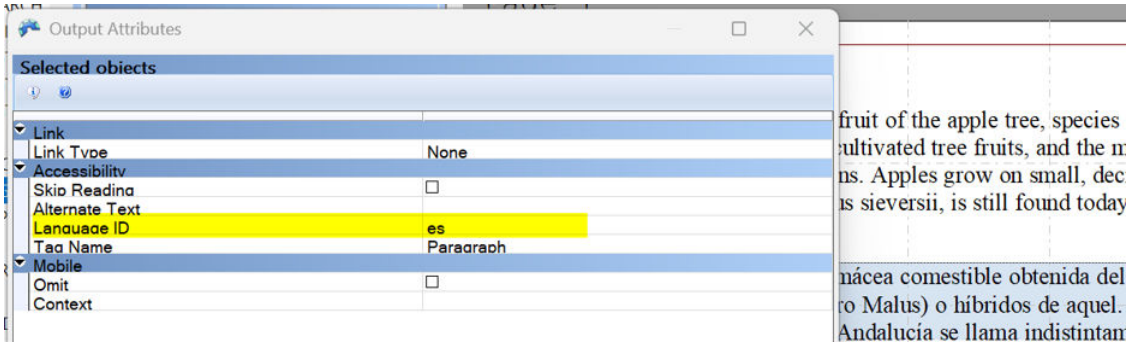
For example, If the user has a text area that uses a different language than the rest of the document. User can specify the language for the entire text area in the text area properties.



User can also specify a language for specific text within a text area. When editing the text area, user can select the text and then right click on the mouse to open the menu and select Edit / Output Attributes.



In the below example, we are setting the highlighted text to use a Language ID of es (Spanish).



2-character ISO 639 language codes are commonly represented using lowercase letters and ISO-3166 country codes are commonly represented using uppercase letters.

User can find a list of ISO-639 Language Codes and ISO-3166 Country Codes at https://docs.oracle.com/cd/E13214_01/wli/docs92/xref/xqisocodes.html.

Setting the Language ID Through DAL

The SetLangID (DAL) function is used to specify a language ID on a text label or variable field.

The SetLangID DAL function uses the following syntax:
Syntax:

```
SetLangID(ObjectName,Section,Form,Key2,ObjectType, LangID)
```

Parameter	Description
ObjectName	Enter the name of the text label or field to set with an ADA tag enumeration. If all naming parameters are omitted, this parameter defaults to the current field. The system updates the first object found that matches your entry for this parameter.
Section	Optional. Enter the name of the section.
Form	Optional. Enter the name of the form.
Key2	Optional. Enter the name of the Key2 group.
ObjectType	Enter the type of object, such as "FIELD" or "TEXT". **Note** The default is "FIELD"
LangID	The Language ID to use for the specified text label or variable

This function optionally returns one (1) if successful or zero (0) if no object was changed.

> **Note**
>
> - The object referenced by SetLangID must have an explicit name.
> - Language IDs are only used when producing accessible PDF output
> - There is no validation of the Language ID to ensure it is a valid ISO code.

Parameter	Description
SetLangID ("MyField", , , , ,"fr")	Search for the field named `MyField` on the current form section and if found, set the associated Language ID to `fr` (French language). Note that the neglecting of the `section`, `form`, and `key2` names means that the search is limited to the currently `active` section. Neglecting of the ObjectType parameter means that `FIELD` is assumed.
SetLangID ("MyLabel", "*", "*", "*", "TEXT", "es")	Search for the first occurrence of a text label with the name `MyLabel` found within the entire transaction and set the Language ID to `es` (Spanish language). Note the the asterisk parameters ("*") means to include all within the search. Using asterisk for `section`, `form`, and `key2` indicates to search the entire transaction.
SetLangID ("AField", "ASection", "AForm", "AKey2", "FIELD", "de")	Indicates to locate a given field on a specific section, form, and key2 grouping within the entire transaction and set the Language ID to `de` (German language).

Updated Form and Section Accessibility Reports

Form and Section Accessibility Reports have been updated to include the Language ID.

The Accessibility Report will display the Language ID for variable fields, text labels, and text areas (including specific text within a text area).

Section Accessibility, ADA Tag Report

Source: D:\fap\mstrres\DMStudio\RPEX1New\FORMS\Languages.FAP

Review: FOUND ACCESSIBILTY DATA

Fields

Name	Tag name (Font)	Options	Alternate Text	Language ID
FIELD	N/A	N/A	N/A	en-US

Text Labels

Name	Tag name (Font)	Options	Alternate Text	Language ID
English (en-US), Spanish (es), French (fr), German (de-DE), Portuguese (pt-pt) text	Paragraph(Times-Roman 10 PT)	N/A	N/A	N/A

Text Areas

Name	Tag name (Font)	Options	Alternate Text	Language ID
TEXTAREA #002	N/A	N/A	N/A	es
La	Paragraph(Times-Roman 10 PT)	N/A	N/A	fr
pomme	Paragraph(Times-Roman 10 PT)	N/A	N/A	fr
est le fruit du pommier, arbre fruitier largement cultivé. L'arboriculture fruitière est une branche	Paragraph(Times-Roman 10 PT)	N/A	N/A	fr
de l'arboriculture spécialisée dans la culture des arbres fruitiers afin d'en récolter les fruits ou les faux-fruits	Paragraph(Times-Roman 10 PT)	N/A	N/A	fr

13

Transactions Importing XML with HTML Markup for MLT Fields

Prior to this version, any markup that might have been included in your XML destined for a MLT field was being ignored. If you had this situation, the output will now translate the character entity correctly.

This is related to a new feature improving the HTML markup support for MLT field mapping. You can find our more about this feature by reviewing the release document for this version.

14

Transactions Using XML Import Rules

When using an XML import rule to load a transaction’s data, sections that are set as “Can Grow” (dynamic), but also specifying a non-custom paper-size (e.g. Letter, Legal, A4, etc.) may now result in change if the section contains any multiline text (MLT) fields.

Combining the Can Grow and Shrink attribute with a fixed (non-custom) paper size is not permitted in the standard DMStudio managed documents. However, if the original form was imported from a legacy (form.dat) style resource definition, it is possible the conflicting attributes might have been defined manually.

The xml import rule was changing the size of the section if the dynamic option was specified, with the theory being that reformatting a text area or MLT that expands could “grow” the section. Since a fixed paper size section cannot technically grow, this causes a formatting conflict and unnecessary pages may get created.

If this impacts your regressions, there are few possible methods to correct.

- If you do not expect to use custom size sections on import, you could try to specify this INI option and see if the problem is resolved.

  ```
  <XMLIMPORT>
  DontResizeImages=Yes
  ```

  This is a legacy option that when included tells the XML import rule to not change dynamic size sections.

  If you do have some sections that should be dynamic and some that should not be, this solution will not be your choice.

- Alternatively, you could remove the dynamic option from the section within the form. As mentioned, DMStudio does not let you manage this combination of options, therefore to change the option manually requires checking the form out of the library and then manually editing. Look for your section name and just below it will be a DAPOPTIONS node with a VALUE attribute.

  ```
  <SECTION NAME="mysection">
  <DAPOPTIONS VALUE="DW#100600"/>
  ```

  It is important that you only remove the W in this value and do not change any of the other characters.

  Once done, save the file and check the form back into the library from DM Studio as a new revision. Subsequent runs will no longer have the problem.

- As a final possible solution, you could specify the DAPOPTIONS node in your import XML. This too requires the manual step of first checking the form out the library to see what DAPOPTIONS VALUE is specified for your section. Copy that setting, omitting the W as indicated in the previous option.

  Changing in this manner will only impact those transactions using this particular input file. Any other transaction files you use would have to be modified similarly.

Glossary

Index