

WE4AUTHORS Cluster

Project full title: “WEb accessibility FOR web AUTHORing tools producers and communities Cluster”

Call identifier: PAWA-2019

Topic: Preparatory Action: Application of web accessibility requirements in web-authoring tools and platforms by default

D1.2 Mapping of user requirements with accessibility features

Deliverable Id:	D1.2
Deliverable Name:	Mapping of user requirements with accessibility features
Version:	2
Status:	Final
Due date of deliverable:	30 October 2020
Actual submission date:	30 October 2020
Work Package:	1. User requirements and prioritising of accessibility features
Organisation name of lead partner for this deliverable:	Funka
Author(s):	Susanna Laurin (Funka), Johan Kling (Funka), Jakob Hasslöw (Funka), Timo Stollenwerk (Kitkoncept), Steffen Ring (Kitkoncept), Mike Gifford (OpenConcept), Joakim Lindkvist (TinyMCE) David Hansson (SiteVision) Anders Korsvall (SiteVision), Joakim Lindqvist (SiteVision)
Partner(s) contributing:	Funka, Kitkoncept, SiteVision (plus senior advisor OpenConcept and Tiny MCE)

Abstract: The mapping of user requirements with accessibility features builds on the result from D1.1 and discussions among the Cluster partners and senior advisors on a list of accessibility features to be developed, taking market impact, technical justification and potential barriers across platforms into account.

Disclaimer:

WE4Authors Cluster has received funding from the European Commission’s Work Programme for 2019 for Pilot Projects and Preparatory Actions in the field of Communications Networks, Content and Technology under grant agreement No LC-01438281.

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Commission. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

This deliverable contains original unpublished work or work to which the author holds all rights except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of contents

1. Introduction.....	6
1.1. Scope and objective of deliverable.....	6
1.2. Methodology of work.....	6
2. Selection criteria.....	6
2.1. Overall criteria based on previous research.....	6
2.2 Web author demands.....	7
3. Analysis on user needs.....	9
4. Technical implications.....	10
5. Mapping features with user needs.....	11
5.1 ALT-text.....	13
Example: Drupal.....	14
Example: Plone.....	17
Example: SiteVision.....	18
Example: TinyMCE.....	19
Features to be tested.....	20
5.2. Change language.....	20
Example: Drupal.....	20
Feature to be tested.....	23
5.3 Documentation.....	23
Feature to be tested.....	25
5.4 Tables creator.....	26
Example: SiteVision.....	26
Example: Plone.....	28
Feature to be tested.....	31
5.5 Forms editor.....	31
Example: Plone.....	37
Features to be tested.....	38
5.6 Video.....	38
Features to be tested.....	39
5.7 Live testing while authoring.....	39
Example: TinyMCE accessibility checker plugin.....	39
Features to be tested.....	41
5.8 Testing of documents.....	41
Features to be tested.....	43
5.9 Testing the content of pages.....	44
Example: Plone.....	45
Features to be tested.....	45
5.10 Testing the whole website.....	45

Features to be tested.....	46
6 Conclusion.....	47

List of figures

Figure 1: Drupal editor showing alt-text alternatives	15
Figure 2: Drupal, editor view showing boxes to be filled out about alt-text	15
Figure 3: Text explaining configuring image fields in Drupal	16
Figure 4: Drupal: editor view of insert image and required text box.....	17
Figure 5: Drupal, editor view showing insert image error message.....	17
Figure 6: Plone, editor view of how it looks when an image uploads.....	18
Figure 7: Plone/Volto: editor view of how to add images.....	18
Figure 8: Sitevision, editor view showing activate extended accessibility support button	19
Figure 9: Sitevision, editor view on where image description is in red and required	19
Figure 10: TinyMCE, showing image editor and source and description box	20
Figure 11: CKEditor, text editor, toolbar configuration button.....	21
Figure 12: CKEditor, Toolbar configuration button showing language option.....	21
Figure 13: Editor in CKEditor showing title, body and tags option.....	22
Figure 14: CKEditor showing editors language option	23
Figure 15: Drupal: showing highlighted information about web accessibility	24
Figure 16: Drupal, Editor view showing help options with links options	25
Figure 17: Drupal, help option describing user options in text.....	25
Figure 18: Sitevision editor showing how edit table type options	27
Figure 19: Sitevision editor showing table properties settings, different checkboxes	28
Figure 20: Plone/Volto, showing different column options in a table	28
Figure 21: Drupal, editor showing table properties options in different boxes.....	29
Figure 22: Drupal, written code about tables showing	30
Figure 23: Drupal, editor window showing cell properties	30
Figure 24: Drupal, view module and code written in different colours	31
Figure 25: Drupal, webpage showing webform accessibility information	34
Figure 26: Drupal, showing webform options with checkboxes	35
Figure 27: Drupal, webform options visual in different tables and categories.....	35
Figure 28: Editor view showing contact information boxes to fill out	36
Figure 29: Form validation boxes showing required message, with clickable options	36
Figure 30: Form validation, required message question mark option, shown in yellow	37
Figure 31: Volto editor, drop down menu showing different form options	37
Figure 32: Volto editor, form showing different boxes to be filled out	38
Figure 33: Volto editor, showing video player and URL input box.....	39
Figure 34: TinyMCE, editor showing accessibility checker box with warning message	40
Figure 35: TinyMCE, editor showing error message in accessibility checker	40
Figure 36: Acrobat pro accessibility checker showing checkboxes, fails and pass	43
Figure 37: Sitevision, editor showing site settings with checkboxes of accessibility	46
Figure 38: Sitevision, email showing accessibility test information	46

List of tables

Table 1: Selection criteria mapped with features	13
--	----

Executive summary

This report summarises the results of in-depth analysis of user requirements, business logic, technical feasibility, existing solutions and potential impact of built-in accessibility support by default.

The selection criteria cover a wide range of perspectives, from standards and end user relevance to standards and frequency of use. In the selection process, all stakeholder views were taken into account and balanced against the possible implementation across platforms.

Some of the features are solving one specific problem (ALT-texts, Change language), whereas others approach a broader perspective and try to support accessibility in a more general sense (different types of accessibility testing).

The selected features are a mix of automation, support and testing, thus covering all categories of built-in accessibility by default services detected in the pilot project as well as in T1.1 of the current project.

Next steps include prototyping and testing of the selected accessibility features. The testing will consider ease of use and technical implementation possibilities, but also to what extent usage of the features result in improved accessibility for the end users.

1. Introduction

1.1. Scope and objective of deliverable

In this deliverable, the consortium will develop a set of selection criteria to support the prioritisation of accessibility by default features and map them to the user requirements detailed in D1.1.

In this activity, different stakeholder perspectives will be taken into consideration. For example, it will be important to assess to what extent the feature:

- will help end users with disabilities to use the interface independently,
- can support web authors comply with the Web Accessibility Directive,
- has the potential to be implemented in various authoring tools.

If possible, the features should cover all 4 WCAG principles and the 9 user groups of the Web Accessibility Directive. The Cluster members will agree on a prioritised list of the most relevant accessibility features to be created in the project and published in the repository.

The main objective of this deliverable is to make sure the features chosen for the project are relevant and prioritised among the different stakeholders of the entire ecosystem of web authoring.

1.2. Methodology of work

The methodology for mapping user requirements with accessibility features can be described as follows:

- Set up of selection criteria.
- Analysis of the report on user needs (D1.1).
- Discussion with partners and senior advisors around existing features as well as feasibility and technical implications of the proposed features.
- Mapping; suggestion and prioritisation of features to be developed.

2. Selection criteria

When selecting the features to test in this project, several perspectives were taken into account:

- Overall criteria based on previous research.
- Web author demands.
- Variety of end user groups supported.
- Standardisation.
- The business perspective.

2.1. Overall criteria based on previous research

The overall criteria that have been applied to the prioritisation process include:

- Frequency of use of the feature in general.
- Frequency of accessibility failures (based on audits performed by the partners).
- Potential of a built-in accessibility feature to solve or support accessibility problems.

- An existing feature that has proven successful.

2.2 Web author demands

From the web author perspective, accessibility by default features can be categorised into four distinctive user scenarios based on when the feature appears during the publishing phase:

- Fully automatic accessibility.
- Half-automated prompts supporting the authors while publishing.
- Information, instructions, manuals and wizards.
- Accessibility testing.

The accessibility features can also be described as mandatory, supportive or informative. There is a balance between the mandatory and automatic features being perceived as fast and easy, and the complex reality where content creation often becomes a compromise between speed and accuracy.

Based on the user requirements of WP1, described in chapter 3 below, the consortium has concluded that a good mix of automation, support, information and testing is needed to fulfil the demands from web authors.

2.3 Variety of end user groups supported

To maximise impact for persons with disabilities, accessibility by default features should aim to benefit as many different end user needs as possible. For the definition of user needs, the consortium has used the list from the Web Accessibility Directive Implementation Acts:

- usage without vision;
- usage with limited vision;
- usage without perception of colour;
- usage without hearing;
- usage with limited hearing;
- usage without vocal capability;
- usage with limited manipulation or strength;
- the need to minimise photosensitive seizure triggers;
- usage with limited cognition.

2.4 Standardisation

In addition to this, the World Wide Web Consortium, W3C, has published the Authoring Tool Accessibility Guidelines, ATAG, which attempt to set a standard for two perspectives of accessibility:

- make the authoring tools themselves accessible, so that people with disabilities can create web content.
- help authors create more accessible web content — specifically: enable, support, and promote the production of content that conforms to the Web Content Accessibility Guidelines, WCAG.

Some authoring tools use ATAG as inspiration, but since WCAG is the standard referred to in regulations, ATAG has to date rarely been referenced in procurement requirements or similar.

2.5 The business perspective

From the perspective of authoring tool producers, modern authoring tools, as well as the websites and web applications that are created with those tools, face a wide variety of requirements. The authoring tools themselves and the websites or web applications that are created with the tools are supposed to be fast, user friendly, accessible and easy to use.

Typical customer demands when choosing authoring tools can include:

- user friendly (most crucial success factor from a business perspective)
- easy to learn
- easy to teach
 - reduce the amount of training that is necessary
 - widen the range of people that can edit content

Typical requirements for websites those authoring tools help to create and maintain:

- fast
- accessible
- consistent corporate design
- user friendly
- easy to use

When the requirements of both the authoring environment and the resulting websites become too complex, this tends to lead to complex tools. This, in turn, makes them less easy to use and requires the authors to receive more training.

The trade-off between requirements and ease of use can be described with an example, where the challenge can be seen from at least two perspectives:

- If ALT-texts are mandatory to fill in, allowing web authors to drag & drop images will be hard, as required fields are often perceived as annoying.
- If drag & drop of images is a requirement, it becomes hard to prompt authors to write an ALT-text without destroying the ease of use.

The business case for accessibility is sometimes hard to make, as accessibility requirements tend to be seen as making the tool harder to use. This leads to frustrated users and less market share, ultimately making it harder to compete.

Many organisations, especially in the public sector, apply a decentralised approach to content creation. This means that many authors are not professional communication officers and many of them only use the authoring tool occasionally. This approach tends to lead to less accessible content. Therefore, tools that are easy to use and require less training are key to success.

Authoring tool vendors and the open source community might want to have different trade-offs. A system like Wordpress, which started as a simple blog platform and still targets small to medium-size websites, will most likely not integrate accessibility enhancements into its core if that makes the

system less user friendly or harder to use. A system such as Plone or SiteVision, which mainly targets large enterprises and organisations where accessibility is not a feature but a basic requirement, might be willing to sacrifice a bit of user friendliness to accessibility. However, the goal must be to create features that enhance accessibility without making life harder for the authors.

Open source authoring tools are always a compromise between different user groups and use cases. Plone, for instance, is used as a CMS for websites as well as a base for corporate intranets and extranets. Even though Plone is mainly used in large corporations, the public sector, universities and research institutions, there are people who use Plone for small to medium-size websites. Some members of the open source community even advocate competing with general purpose systems like Wordpress who target a completely different market than Plone. Therefore, every open source community needs to decide how focused they want to be on a specific market or target audience.

The same is true for commercial vendors of licensed authoring tools. Accessibility and inclusive design do broaden the audience and are mandatory requirements when selling to public sector agencies in the EU covered by the Web Accessibility Directive - or organisations covered by similar legislation elsewhere. If the target audience does not care much about accessibility, it is even more important to make sure that the built-in features are not seen as an obstacle. For some authoring tool producers, the solution may be to provide two different versions of the same tool. However, the aim of this project is, of course, to prove that there doesn't have to be a conflict between ease of use and accessibility support.

It is an important part of the selection process in the project to look at specific use cases and try to come up with accessible, inclusive solutions that make the authoring tools useful and user friendly, whilst at the same time supporting the authors in creating accessible content.

3. Analysis on user needs

Based on the requirements of different stakeholders presented in the Report on user needs (D1.1), it is clear that accessibility features can support web authors in a variety of ways, depending on the situation, previous knowledge and personal preferences.

For web authors with limited experience in publishing – generally, as well as in an accessible way - full automation may be the most efficient support. As this is only possible for basic accessibility features, it will not solve all problems.

Most stakeholders suggest contextual support in one way or the other. It is important that this feature (that can contain many different parts and cover many different topics) provides the user with support on the go.

Instructions, documentation, manuals, handbooks, wizards and information are demanded by many different stakeholder groups. One might have thought that this kind of help would be most valuable for professional web authors who have the time and willingness to find and read instructions to achieve a higher level of accessibility. However, since all user groups have asked for it, it is probably beneficial to all types of users publishing online.

Testing and validation are other recurring items on different levels. Depending on where and when the testing is performed, the user benefiting most from it may be the author or the responsible website owner. If validation before publication and mandatory accessibility fields were abundant, the digital world would of course look very different.

Three items are recurring in all input from web authors:

- documents
- tables
- forms

They seem to be among the most difficult areas of content creation when it comes to accessibility.

Lack of knowledge and awareness are seen as the biggest reasons behind low levels of accessibility, but troublesome handling of tools and cumbersome solutions are also mentioned quite often.

4. Technical implications

There is also still no real harmonisation on legal accessibility requirements between different markets, as the EU follows the minimum requirements of WCAG 2.1 AA, but the U.S. and Norway are still on 2.0 AA (with exceptions). On the other hand, the legislation of both the U.S. and Norway covers the private as well as public sectors, whereas the Web Accessibility Directive only covers bodies covered by public law. When the European Accessibility Act enters into force in 2025, the risk is that this situation will be even more complex.

As most authoring tools strive to meet the demands of several geographical markets and sectors, they often provide a base configuration in their core that is a compromise between the different requirements and target audiences. The wider the scope, the more compromises an authoring tool has to make.

One option to solve this problem is to factor out specific requirements such as accessibility into an add-on product. Technically speaking, it is non-trivial to allow add-on products to amend the behaviour of the core. Especially when it comes to usability and user experience (UX) issues. For user friendliness and ease of use, consistent behaviour of the system is the key. Add-on products often cannot (and should not) change the core system in such a fundamental way. E.g. if you show inline hints for improving accessibility, this should happen elsewhere in the system in a similar way. If you show them in a different way, authors might become confused.

Getting functionality into the core of a system is often a “political” question. In open source communities, people have different opinions about the main target group of the system. This is often driven by business needs. A company that is mostly active in the public sector will happily welcome accessibility enhancements in the core, while a company that targets hobby users may not be willing to accept a change in the behaviour of the system for that goal.

It is possible to make certain features optional within the core. E.g. the multilingual enhancements of authoring tools such as Plone and Drupal are shipped within the core but need to be enabled by the administrators. Even though both Plone and Drupal target large and often multilingual website

and intranet use cases, some projects do not require multilingual functionality. Showing the options to translate content or to switch between languages may not be helping but rather confuse web authors if they are not relevant to their usage. Building a user-friendly authoring tool often means reducing the options in the user interface and focusing on the parts that matter for that particular author.

Therefore, while making features optional or configurable in the core is a good solution for some use cases, it is not a good solution for all use cases. For example, if a system has hundreds of settings, it may overwhelm both administrators and web authors. Too many options also make it difficult to properly test the possible options and make sure they work well together. This is also true for add-ons. Many authoring tools on the market severely suffer from the number of add-on products because they are often incompatible with each other, outdated or pose a severe security risk.

When deciding if a feature is better in core, as an option, or as an add-on product, several perspectives have to be taken into account. The answer highly depends on the underlying architecture of the authoring system as well as on the target groups of the system. Such a decision needs to take into account the system architecture, security architecture, UX/UI questions, accessibility, speed, community decision making and long-term maintenance trade-offs.

An important part of this project will be to make those trade-offs for the given specific accessibility use cases and document the reasoning behind the decisions that will be made to support the wider uptake of the features in the market.

5. Mapping features with user needs

Based on the agreed selection criteria and the analysis of Report on user needs (D1.1), structured discussions with Cluster partners and senior advisors around existing features as well as feasibility and technical implications of the proposed features were held. Each Cluster member presented its experience and preference in a series of facilitated online meetings. During the meetings, notes were taken and distributed among partners. Between meetings, technical documentation, examples and screenshots were provided in co-created documents. The consortium also started a Discord (an instant messaging application) channel to be able to share thoughts and ideas in an efficient way.

As a result of the fruitful discussions, the consortium agreed on ten features, which are deemed to be the most relevant, useful and feasible features that it is possible to implement in different kinds of authoring tools to support web authors in publishing accessible content.

The ten agreed features to be prototyped and tested are:

- ALT-text
- Change language
- Documentation
- Tables creator
- Forms editor
- Video
- Testing in editor
- Testing of documents

- Testing of pages
- Testing of the whole website

The mapping of features to selection criteria is shown in the following table:

Selected Features	Overall criteria based on previous research	Web author demands	Variety of end user groups supported	Standardisation	The business perspective
5.1: ALT-text	Frequently used	Perceived as resource demanding	Usage without vision	WCAG 2.1 AA 1.1.1	Non mandatory
5.2: Change Language	Potential to solve problem	Important to users of assistive technology reading content out loud	Usage without vision, Usage with limited cognition	WCAG 2.1 AA 3.1.2	Optional
5.3: Documentation	Has proven successful	Lack of knowledge is a recurring item in stakeholder input	All user groups	ATAG 2.0 B.4.2	Standard
5.4: Tables creator	Frequency of accessibility fails	Top 3 demand in survey	Usage with limited manipulation or strength, Usage without vision, Usage with limited cognition	WCAG 1.3.1	May come at different levels for different markets/sectors
5.5: Forms editor	Frequency of accessibility fails	Top 3 demand in survey	Usage with limited cognition, Usage without vision, Usage with limited manipulation or strength,	WCAG 2.1 AA 1.3.5, 1.4.1, 1.4.10, 1.4.11, 1.4.13, 2.1.1, 2.1.2, 2.4.3, 2.4.4, 2.4.6, 2.4.7, 2.5.1, 2.5.2, 2.5.3, 3.2.1, 3.2.2, 3.2.3, 3.2.4,	May come at different levels for different markets/sectors

				3.3.1, 3.3.2, 3.3.3, 3.3.4, 4.1.1, 4.1.2, 4.1.3	
5.6: Video	Has proven successful	Perceived as resource demanding	Usage without hearing, Usage with limited hearing, Usage without vision	WCAG 2.1 AA 1.2.2, 1.2.3, 1.2.4	Non mandatory
5.7: Testing in editor	Potential to solve problem	High demand for testing	All user groups	ATAG 2.1 B.3.1	User friendly
5.8: Testing of documents	Potential to solve problem	Top 3 demand in survey	All user groups	ATAG 2.1 B.3.1	User friendly
5.9: Testing pages	Has proven successful	High demand for testing	All user groups	ATAG 2.1 B.3.1	User friendly
5.10: Testing of the whole website	Has proven successful	High demand for testing	All user groups	ATAG 2.1 B.3.1	User friendly

Table 1: Selection criteria mapped with features

Below, an in-depth description of each of the selected features is provided. The description includes the end user perspective, existing functionality on the market and chosen set up for prototyping and testing for each of the features.

5.1 ALT-text

An ALT-text means Alternative text, a description of a visual element in text. ALT-texts are important for visually impaired users with screen readers, who can only understand the purpose of the image when it is described in text that the assistive technology can convert into read out wording or braille.

For web authors, every image published requires an active decision; should this image be described or not? If it needs a description, what should I write? There is ongoing research¹ where AI is used to describe the content of the image in words, but until this research results in market solutions, ALT-texts require both awareness and manual labour.

ALT-text is one of the most basic elements of achieving WCAG compliance, yet it is regularly missing or inappropriately defined by authors. WCAG 2.1 Guideline 1.1 states that a site must “provide text

¹ <https://azure.microsoft.com/en-au/services/cognitive-services/computer-vision/>

alternatives for any non-text content". In the WebAim Million study of the top million websites, "31.3% of all home page images (12 per page on average) had missing alternative text".²

[WebAim Million: a study of the top million websites](#)

As with most automated tools, this survey couldn't determine if the ALT-text that was provided was meaningful or not. It isn't uncommon to see useless ALT-text with uninformative content, such as the filename or "image".

Most authoring tools support adding alternative text, but very few go beyond this. Clearly having the ability for the author to add ALT-text in the User Interface is a good first step.

In the project survey, participants were asked if they thought that the authoring tool should:

- Inform me (a)
- Actively prompt me (b)
- Make accessibility mandatory (c)

Both (a) and (b) provide a clear benefit in reminding users about the importance of alternative text. Option (c) - making ALT-text mandatory is problematic as there are some instances where it isn't appropriate.

There are cases where an image is clearly decorative and should be indicated with alt="". A duplicated image (i.e. 5 hearts in a row), should only have alt text on the first image (with the rest being decorative).

This can be seen to be part of ATAG 2.0 B.2.3: Assist authors with managing alternative content for non-text content³ and B4.1.1 Features Active by Default.⁴

[ATAG 2.0 B.2.3: Assist authors with managing alternative content for non-text content](#)

[B.4.1.1 Features Active by Default.](#)

This could be further enhanced by ensuring that users are providing useful content. ALT-text which is simply "image", "photo", or includes "*.jpg" should also result in a warning.

The authoring tools handle the issue of ALT-texts in different ways:

Example: Drupal

The Drupal community chose a different route with Drupal 8, where the ALT-text is mandatory, but can be overridden. The philosophy behind this was to make it as difficult to not enter ALT-text than it was to write it. The Drupal community wanted to influence user behaviour to support a more accessible pattern.

² <https://webaim.org/projects/million/#alttext>

³ https://www.w3.org/TR/2015/NOTE-IMPLEMENTING-ATAG20-20150924/#gl_b23

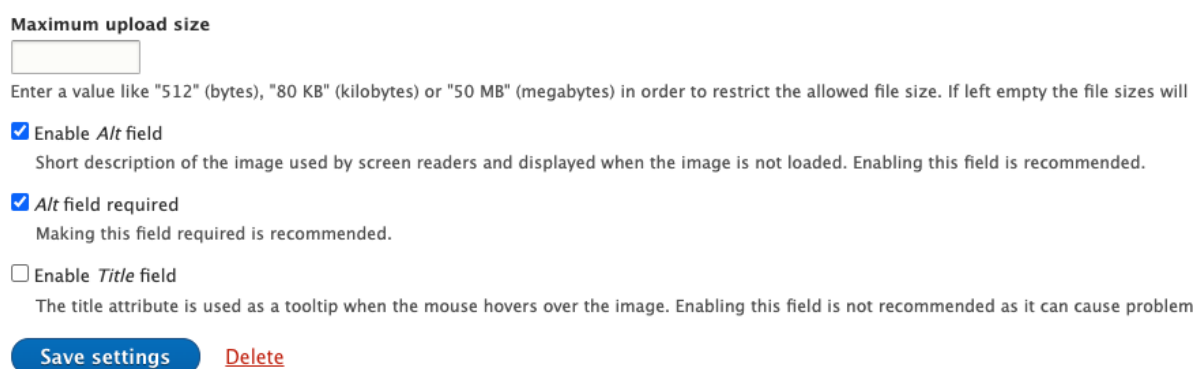
⁴ https://www.w3.org/TR/2015/NOTE-IMPLEMENTING-ATAG20-20150924/#sc_b411

There hasn't been a study to see if it has worked for ALT-text, but there is some evidence from the WebAIM Million project that Drupal sites are getting more accessible over time.

With Drupal 8 there were two main areas where images are added, Image Fields and CKEditor.

With the Image fields there was no clear argument as to why a null ALT-text would be appropriate, so this was set to be required by default.⁵ This was just setting the default condition when adding an Image field to a form or Content Type. The developer can always override the accessibility defaults when creating or editing a Content Type.

[Drupal: Make the default 'alt' attribute for Image fields required](#)



Maximum upload size

Enter a value like "512" (bytes), "80 KB" (kilobytes) or "50 MB" (megabytes) in order to restrict the allowed file size. If left empty the file sizes will

Enable *Alt* field
Short description of the image used by screen readers and displayed when the image is not loaded. Enabling this field is recommended.

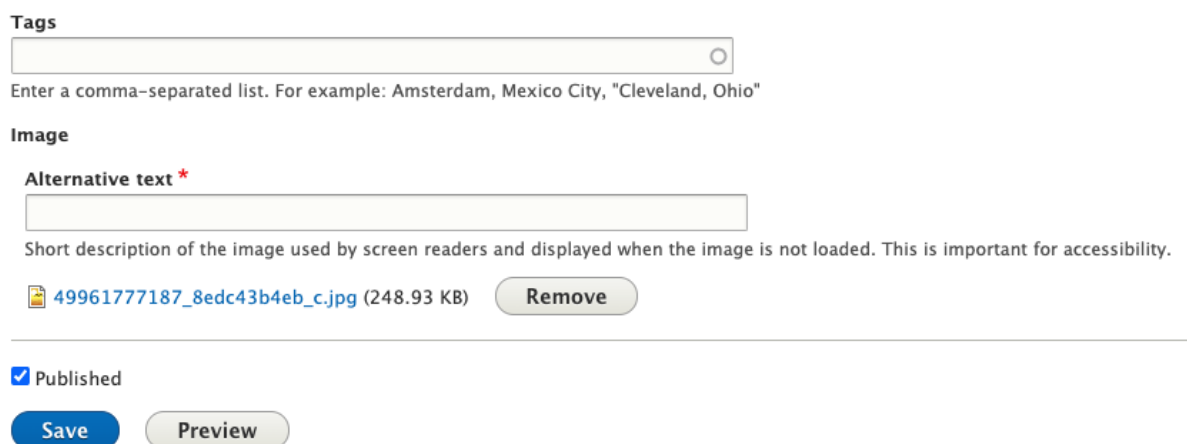
***Alt* field required**
Making this field required is recommended.

Enable *Title* field
The title attribute is used as a tooltip when the mouse hovers over the image. Enabling this field is not recommended as it can cause problem

[Save settings](#) [Delete](#)

Figure 1: Drupal editor showing alt-text alternatives

Default Image Field Form:




Tags

Enter a comma-separated list. For example: Amsterdam, Mexico City, "Cleveland, Ohio"

Image

Alternative text *

Short description of the image used by screen readers and displayed when the image is not loaded. This is important for accessibility.

 49961777187_8edc43b4eb_c.jpg (248.93 KB) [Remove](#)

Published

[Save](#) [Preview](#)

Figure 2: Drupal, editor view showing boxes to be filled out about alt-text

⁵ <https://www.drupal.org/project/drupal/issues/2303765>

In striving to meet ATAG 2.0 requirements it was also important to record this in the documentation that an author would access.⁶

[Drupal: Document accessibility features in Image](#)

Configuring image fields

A few of the settings for image fields are defined once when you create the field and cannot be changed later; these include the choice of public or private file storage and the number of images that can be stored in the field. The rest of the settings can be edited later; these settings include the field label, help text, allowed file extensions, image resolution restrictions, and the subdirectory in the public or private file storage where the images will be stored. The editable settings can also have different values for different entity sub-types; for instance, if your image field is used on both Page and Article content types, you can store the files in a different subdirectory for the two content types.

For **accessibility** and search engine optimization, all images that convey meaning on web sites should have alternate text. Drupal also allows entry of title text for images, but it can lead to confusion for screen reader users and its use is not recommended. Image fields can be configured so that alternate and title text fields are enabled or disabled; if enabled, the fields can be set to be required. The recommended setting is to enable and require alternate text and disable title text.

Figure 3: Text explaining configuring image fields in Drupal

WYSIWYG

Because CKEditor allows for less structured data, the Drupal Community needed the author to be able to override the required alt text.⁷ This was also documented in the internal help pages.⁸ When images are added through CKEditor, an accessible modal window pops up which requires the user to enter ALT-text:

[Drupal: Make the "alt" attribute required in EditorImageDialog](#)

[Drupal: Document accessibility features in CKEditor](#)

⁶ <https://www.drupal.org/project/drupal/issues/2308549>

⁷ <https://www.drupal.org/project/drupal/issues/2297681>

⁸ <https://www.drupal.org/project/drupal/issues/2308515>

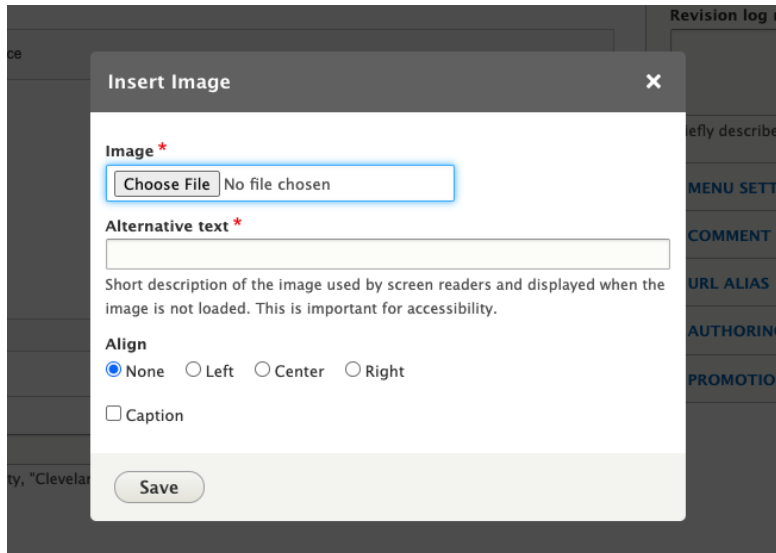


Figure 4: Drupal: editor view of insert image and required text box

If they don't enter the ALT-Text, the error message encourages them to do this, but provides a way for them to override it by entering double quotes ("").

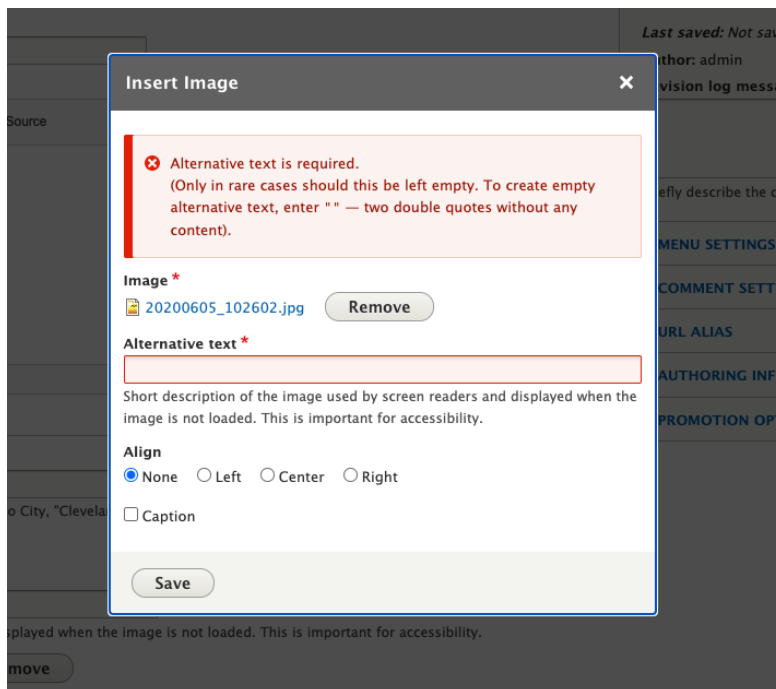


Figure 5: Drupal, editor view showing insert image error message

Example: Plone

The two main ways to add images in Plone are through the WYSIWYG editor & through content types.

WYSIWYG

Plone 6 / Volto provides an ALT-Text field for the image content type. This field is automatically filled with the title of the uploaded image file, when a new image content object is created.

As decorative images should have empty ALT-Texts, the image field is not required.⁹ This wouldn't be possible any longer if the ALT-Text field would be required.

[W3C: how to use the alt attribute of the element in various situations.](#)

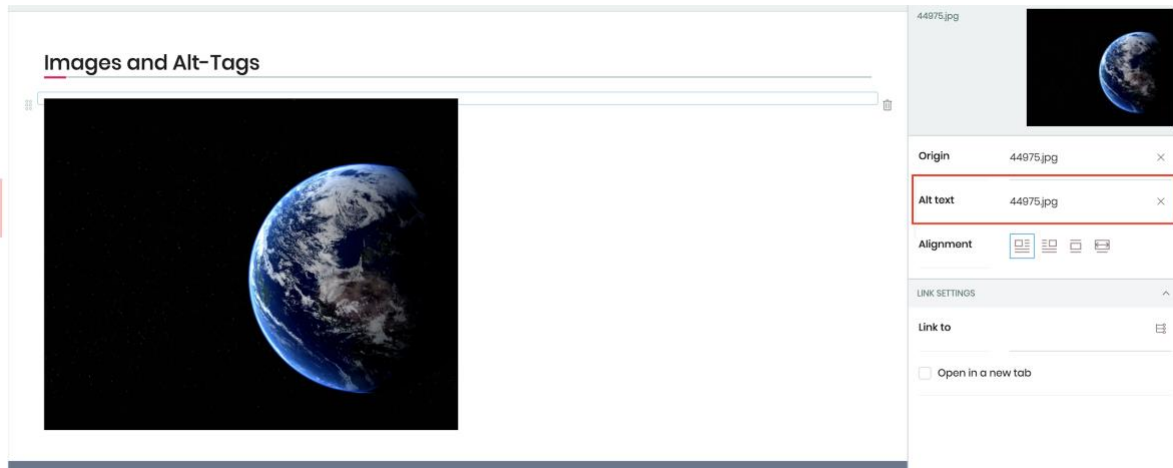


Figure 6: Plone, editor view of how it looks when an image uploads

The downside of this approach is that the web author is not really aware that the ALT-Text should be filled out with a proper description. Plone is considering removing the auto-fill functionality and to replace it with an inline hint for the web authors to fill out the ALT-Text.

Image content type

Plone/Volto additionally has the option to add images on the website and save them inside the folder structure with the image content type. These images can be reused and incorporated anywhere in the website.

Plone currently does not provide an ALT-Text field for the image content type.

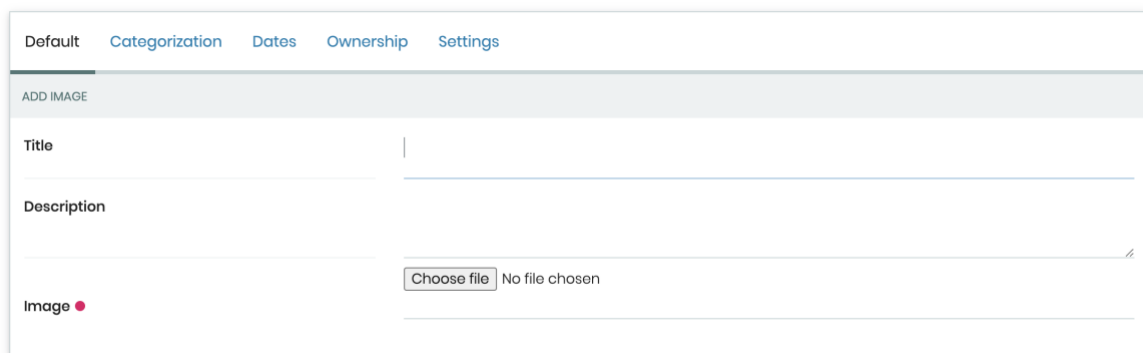


Figure 7: Plone/Volto: editor view of how to add images

Example: SiteVision

In SiteVision the ALT-text is preset as mandatory, when the accessibility function “activate extended accessibility support” is preselected.

⁹ <https://www.w3.org/WAI/tutorials/images/decision-tree/>

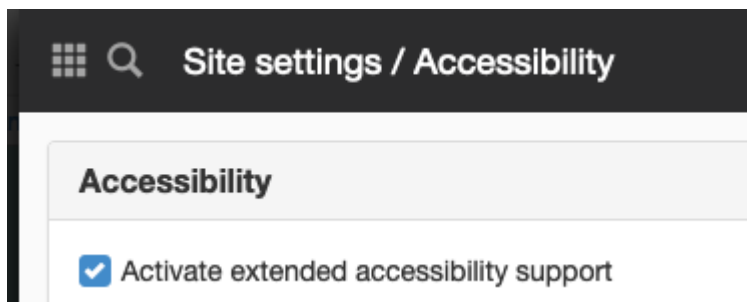


Figure 8: Sitevision, editor view showing activate extended accessibility support button

The extended accessibility support requires an ALT-text on all images and will not allow web authors to add images without actively choosing one of the options described below.

The web author can:

- choose to add an ALT-Text manually.
- use the metadata description as ALT-text.
- select no ALT-Text in case of decorations etc.

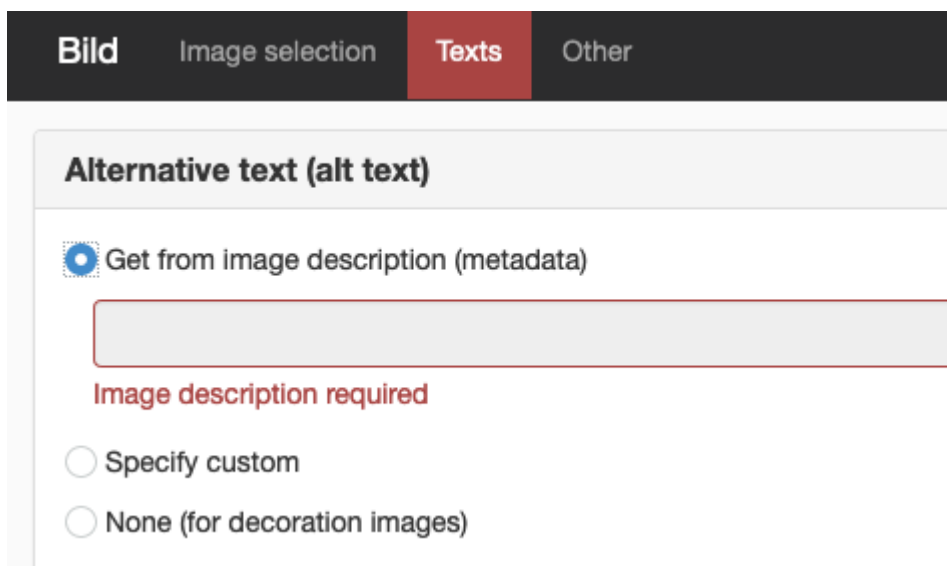


Figure 9: Sitevision, editor view on where image description is in red and required

Example: TinyMCE

Available for purchase as a plugin to TinyMCE is a tool for checking the accessibility of the created document which checks the document against the WCAG 2.1 AA standards (where applicable). The plugin can be added by CMS vendors, CMS implementers or site owners if they are running a standard TinyMCE version.

The commercial plugin available for TinyMCE has a check for ALT-Text built in. The image functionality in TinyMCE allows you to set the ALT-Text when adding the image.

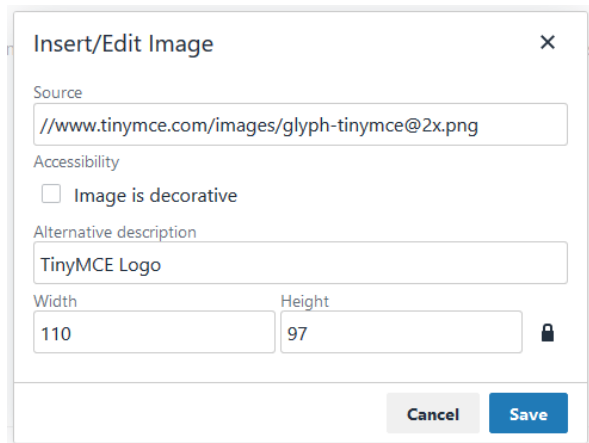


Figure 10: TinyMCE, showing image editor and source and description box

This configuration also has `ai11y_advanced_options: true` that will disable the ALT-Text field in the dialog and prevent the accessibility checker to skip checking that image for an ALT-Text. This was a requirement that came in from the Open Source community. The accessibility checker is described below.

Features to be tested

The ALT-Text issue is going to be prototyped and tested in several ways:

- mandatory vs possible to skip.
- semi-automatic vs manual.
- support vs information.
- examples and documentation.

5.2. Change language

The change language feature provides a way to choose the language of the page or paragraph so that it can be read out aloud with the correct pronunciation (i.e. using the correct speech engine).

The feature is important to users of assistive technology reading content out aloud, i.e. different kinds of screen readers, aimed for both visually impaired and reading and writing impairments.

In WCAG 2.1 SC 3.1.2 Language of Parts (Level AA): The human language of each passage or phrase in the content needs to be programmatically determined. It is quite common to have pieces of text in the body of a page which is different from the main language of the page. Authors often have the responsibility for introducing these through the WYSIWYG editor of the authoring tool.

Example: Drupal

In Drupal 8, a language toolbar button was added, so that it would be possible to easily select an alternative language for some or all of the content within the WYSIWYG.¹⁰ However, it is not there by default but needs to be added in the configuration:

¹⁰ <https://www.drupal.org/project/drupal/issues/1993928>

Drupal: Introduce a language toolbar button

Text editor

CKEditor ▾

TOOLBAR CONFIGURATION
Move a button into the *Active toolbar* to enable it, or into the list of *Available buttons* to disable it. Button toolbar groups will be removed upon save.

Available buttons

U B I S x² x₂ I_x [list icons] [undo] [redo] [cut] [paste] [link] [unlink] [Ω] [undo] [redo] [話] Styles ▾

Active toolbar

B I S x² x₂ I_x [link] [unlink] [list icons] [quote] [image] [table] [list icons] Format ▾ [undo] [redo]

CKEditor plugin settings

Figure 11: CKEditor, text editor, toolbar configuration button.

Once it is selected, it needs to be configured to either allow just the official UN languages, or the “extended list will show all 95 languages that are available in Drupal.” It would be useful if this were extended so that you could select the list of languages that an author could choose from.

Text editor

CKEditor ▾

TOOLBAR CONFIGURATION
Move a button into the *Active toolbar* to enable it, or into the list of *Available buttons* to disable it. Buttons may be moved with the mouse or keyboard arrow toolbar groups will be removed upon save.

Available buttons

U B I S x² x₂ I_x [list icons] [undo] [redo] [cut] [paste] [link] [unlink] [Ω] [undo] [redo] Styles ▾

Active toolbar

B I S x² x₂ I_x [link] [unlink] [list icons] [quote] [image] [table] [list icons] Format ▾ [話] [undo] [redo]

CKEditor plugin settings

Image Uploads enabled, max size: 2 MB	United Nations' official languages ▾ The list of languages to show in the language dropdown. The basic list will only show the six official languages of the UN .
Language	

Enabled filters

Figure 12: CKEditor, Toolbar configuration button showing language option

When that has been configured, the author will see the following button in CKEditor:

Title *

Body (Edit summary)

B *I* ~~S~~ ^{x²} _{x₂} I_x | | | | Format ▾ | Source

Text format Full HTML ▾ [About text formats ?](#)

Tags

Enter a comma-separated list. For example: Amsterdam, Mexico City, "Cleveland, Ohio"

Figure 13: Editor in CKEditor showing title, body and tags option

If the author wanted to add text like this:

```
<p>This paragraph is in English, but...<br />
<span lang="es">Este apartado está en español.</span></p>
```

Then there would be a simple dropdown which allows for the Spanish language text to be properly identified. It is also worthwhile pointing out that the ISO language code is displayed if the text is available in another language.

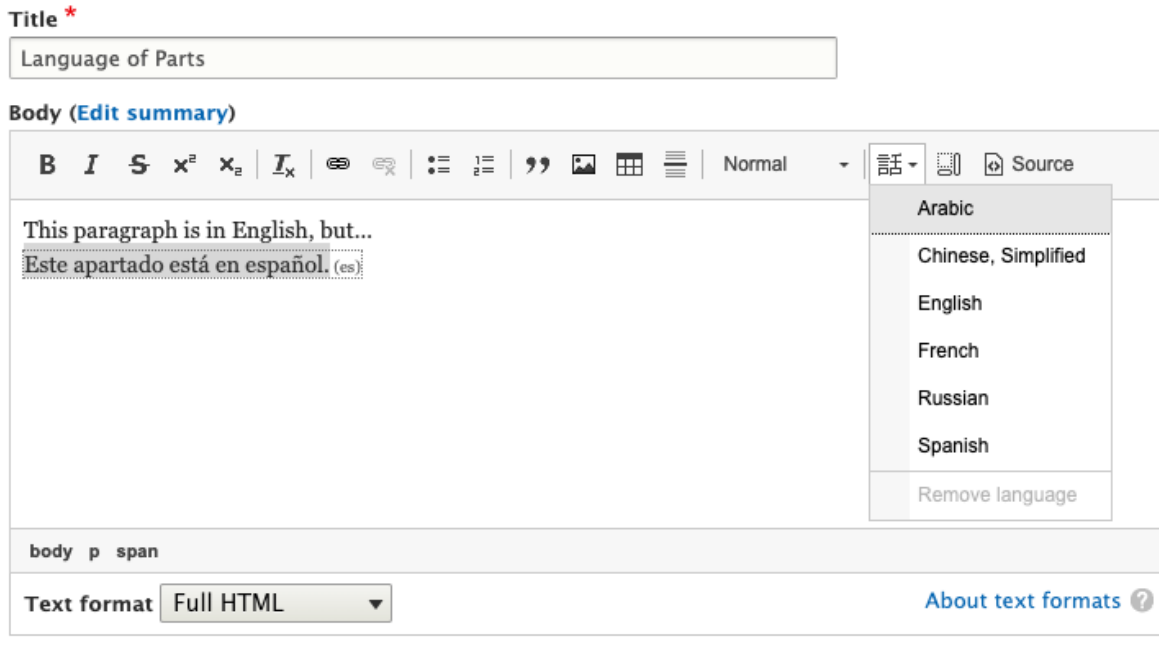


Figure 14: CKEditor showing editors language option

Many of the accessibility features of CKEditor are outlined in the Drupal 8 in-site help, but unfortunately the language toolbar button isn't one of them. There is now an issue to help rectify that in Drupal 9.1.¹¹

[Drupal: Add a description for the language toolbar button to the CKEditor help page](#)

Feature to be tested

The language toolbar button will be prototyped and tested to balance the ease of use/not too many options with the need for the change language feature to be easily found. As not all authors or organisations work in multilanguage environments, the feature may not be as relevant in all use cases. Therefore, the focus of the testing will be:

- Placement and priority
- Design and naming

5.3 Documentation

Documentation of accessibility features is important for all authors. However, as the background and previous knowledge of web authors vary a lot, it is difficult to assess what level of detail the documentation should contain.

There are also essentially two ways of providing documentation; in context or separately. Both ways have their pros and cons and most probably, it depends on the web author and the situation.

¹¹ <https://www.drupal.org/project/drupal/issues/3150364>

Example: Drupal

Separate (external) Documentation

On the footer of every page of Drupal.org there is a link to highlighted information about Web Accessibility.

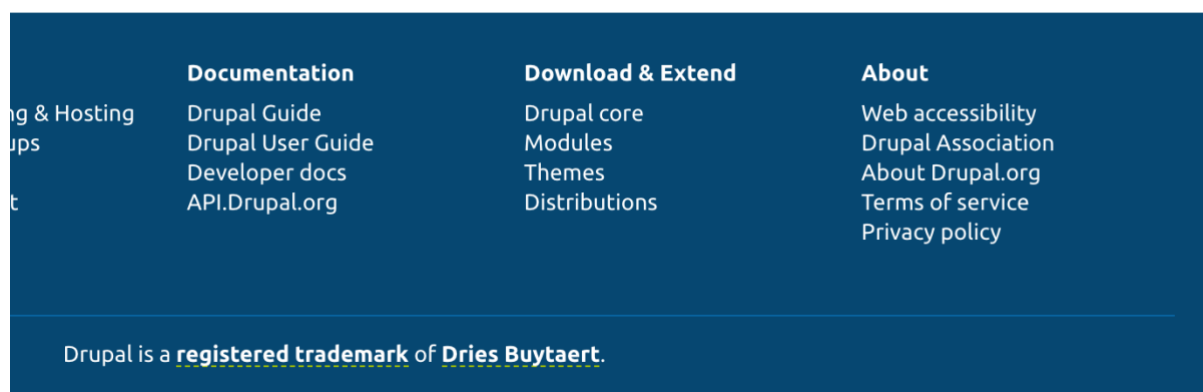


Figure 15: Drupal: showing highlighted information about web accessibility

This link takes you to the central accessibility page¹² which highlights the goals of the community and includes links to documentation of the accessibility best practices for development as well as module and themes development. This page includes the Drupal commitment to WCAG 2.1 AA and ATAG 2.0 AA goals, as well as instructions for where to go to report barriers to the community. The EU is leading the way in the adoption of WCAG 2.1, and other countries are exploring or endorsing EN 301 549. Even in the USA, many are striving to meet the WCAG 2.1 even though the U.S. procurement legislation Section 508 was only recently updated to WCAG 2.0 AA.

[Drupal: Accessibility feature page](#)

Contextual (internal) Documentation

Drupal 8 has a built-in Help module that is structured by feature. This can be disabled, and a website might be structured so that not everyone has access to this. That said, it is structured by feature so that the help material only appears if a module has been enabled. In Drupal 9, administrators are given links to the help screen in the main toolbar and the page looks like this:

¹² <https://www.drupal.org/about/features/accessibility>

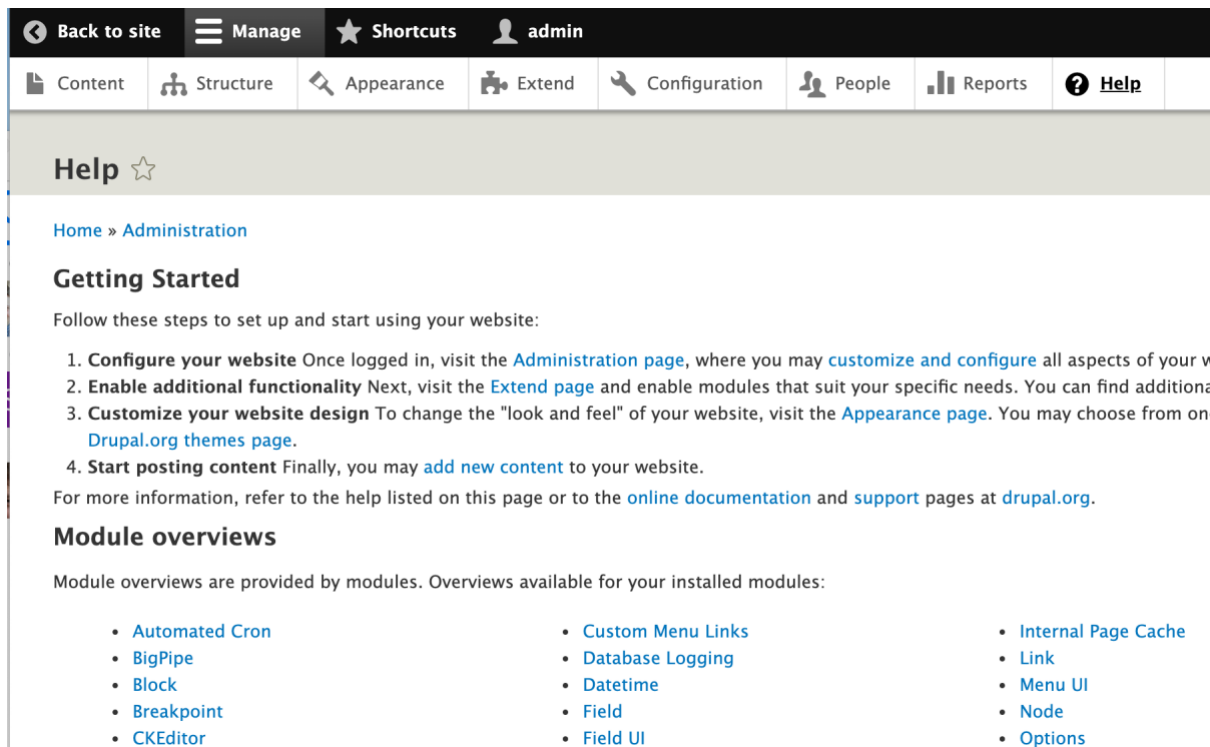


Figure 16: Drupal, Editor view showing help options with links options

A description of the accessibility feature is provided where appropriate in the help text which is available. Often accessibility does require getting an author to do something (like write caption/summary elements) but most authors are not aware it would be helpful.

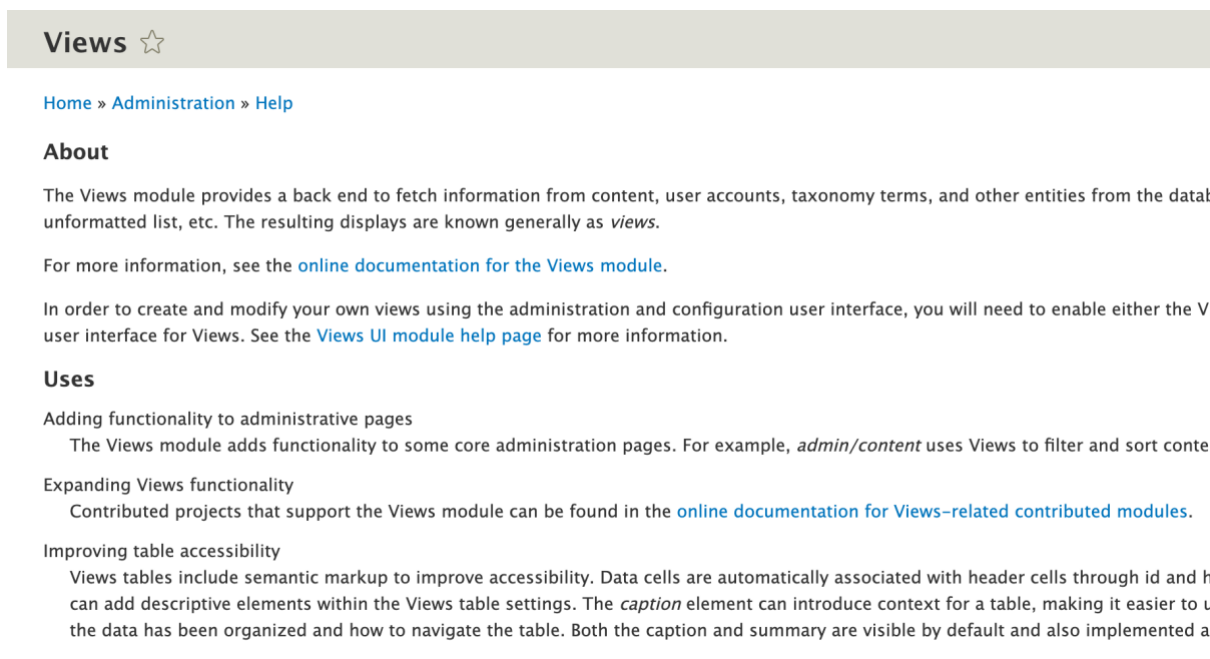


Figure 17: Drupal, help option describing user options in text

Feature to be tested

Documentation will be tested in two dimensions:

- Length and depth (level of detail) of the documentation.
- Contextual help vs separate documentation.

5.4 Tables creator

Tables are used online to show content in a structured way, just as in documents. Many authoring tools provide a special functionality for creating tables to support the web author.

For visually impaired users with screen readers, the accessibility of tables is what makes them possible to understand, as it provides the structure of headings, columns etc. If a table is not created in a correct way, the assistive technology will read the content from left to right, without providing any clue as to how the content is supposed to be understood. For everyone else, the design of the table is important to make it easily readable and understandable.

This is relevant to WCAG Success Criterion 1.3.1 (A) Info and Relationships

An explicit connection between headings and cells allows screen readers to read content in a logical order. The attribute scope on the **th element** presents which data cells the headings belong to. Approved values are row, col, rowgroup and colgroup. The element colgroup will mark the column groups, as tbody will mark grouped rows. One option in expressing the relation between cells is using the id attribute on the th element and the attribute headers on the **td element**. Headers can consist of one or several identities on cell headings.

Example: SiteVision

In SiteVision, the tables are an integrated part of the text-modules, where web authors can create a table using a table template. These templates can be predefined by an administrator of the website. In the table template, headings and rows can be designed with margins /padding, colours and functions related to responsiveness. The template pre-sets the fonts used in the table template and alignments. The web author can change settings, select to use a caption, and also hide it from the display, so that it will not affect the design of the page.

Edit table type
General
Advanced

Editorial information

Name

Description

Default table type

Headings

Content

Margin and borders

Caption

Responsive web

Preview

Abc	Abc	Abc
Abc	Abc	Abc
Abc	Abc	Abc
Abc	Abc	Abc
Abc	Abc	Abc

Cancel

?

Figure 18: Sitevision editor showing how edit table type options

Properties for selected table
Settings
Appearance

Table properties

Table type
 x ▼

Heading type
 ▼

Width
 % ▼ - +

Use caption

Hide the table caption in display mode

Allow visitors to sort in the table

Responsive web

Only show in breakpoints

Hide in breakpoints

Responsive type
 x ▼

OK
Cancel
?

Figure 19: Sitevision editor showing table properties settings, different checkboxes

Example: Plone

Plone 6 / Volto comes with a Table block that allows editors to create tables without the need to master HTML.



Figure 20: Plone/Volto, showing different column options in a table

Web authors can add and delete columns and rows, mark table cells as headers, etc. By restricting the options of how tables can be used, we can control the HTML output and make sure the generated HTML follows accessibility best practices.

The table block in Volto follows the strict HTML rules of tables, which allows screen readers to precisely read out the content that's in the table. The web author can add rows and columns, the content within these is wrapped in a paragraph.

There is no functionality to highlight the first row or column, which often acts like a header cell.

Example: Drupal

WYSIWYG Interface (CKEditor)

Caption and summary elements are available by default when creating a new table.

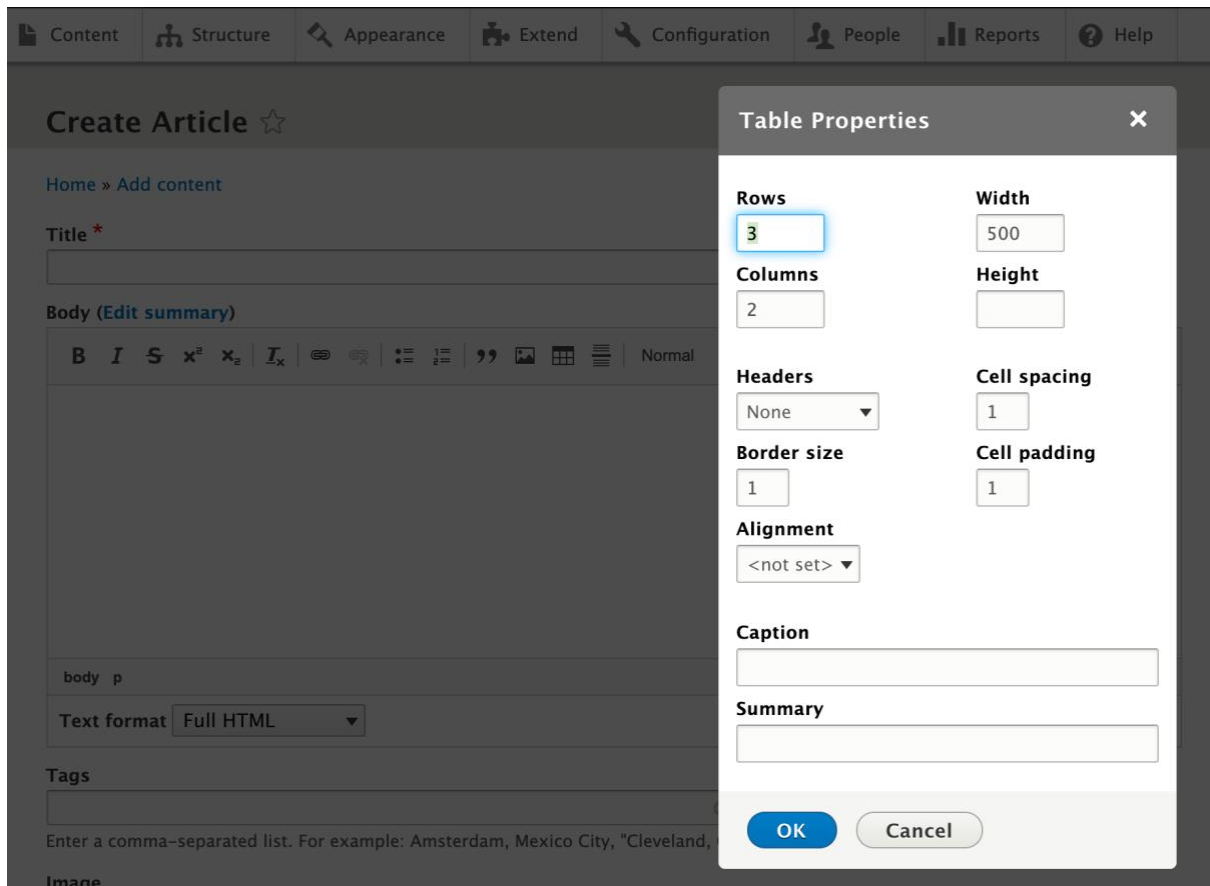


Figure 21: Drupal, editor showing table properties options in different boxes

This produces a structure which is consistent with HTML4's best practices:

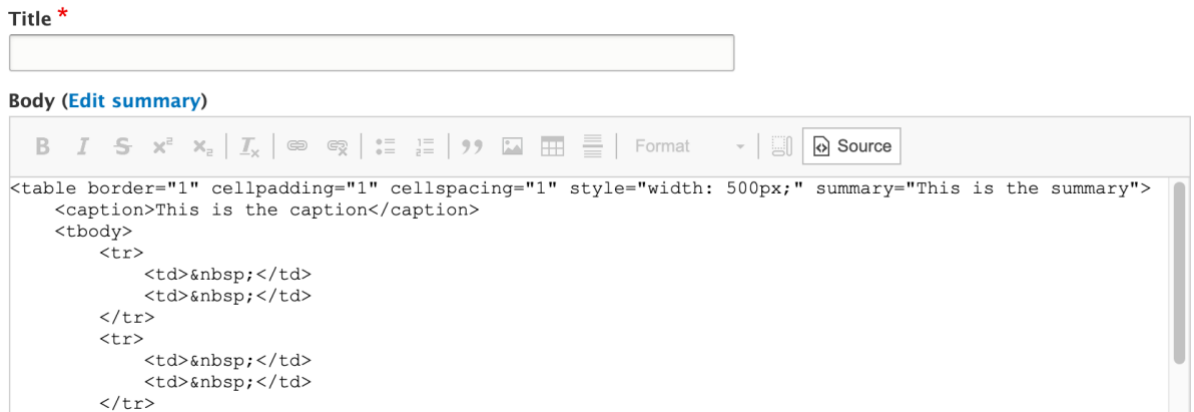


Figure 22: Drupal, written code about tables showing

Headings can be added to individual cells by right clicking in the cell and changing the cell type:

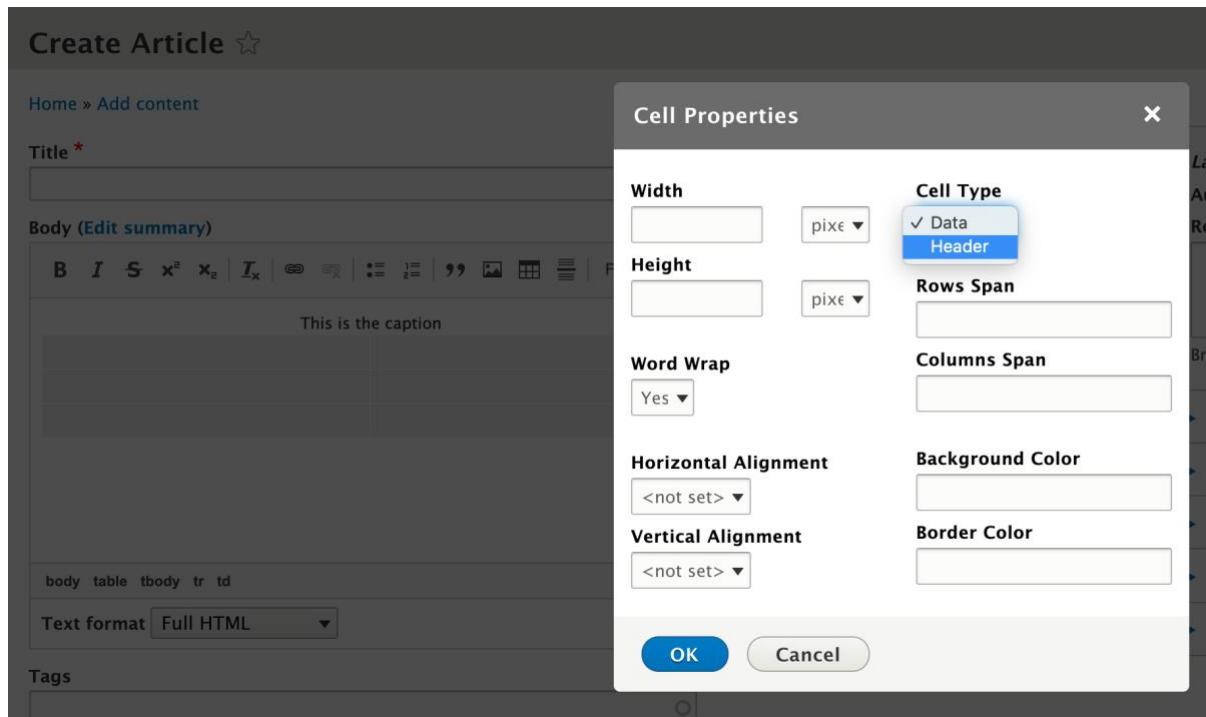


Figure 23: Drupal, editor window showing cell properties

Views

Tabular data is a very popular way to view content in Drupal. The Views module is a query engine which can be used to create tables out of almost any combination of data in the CMS. In Drupal 8, we checked to see that this complex tool was accessible, but that it also incorporated best practices for authors. Views is also used extensively in Drupal’s administration interface presenting lists of users.

By default, the first row of a Drupal View is a header with a unique ID and every cell uses the headers attribute to be associated to the ID of the column it is in. It isn’t possible to create Views in Drupal Core that require a more complicated table relationship.

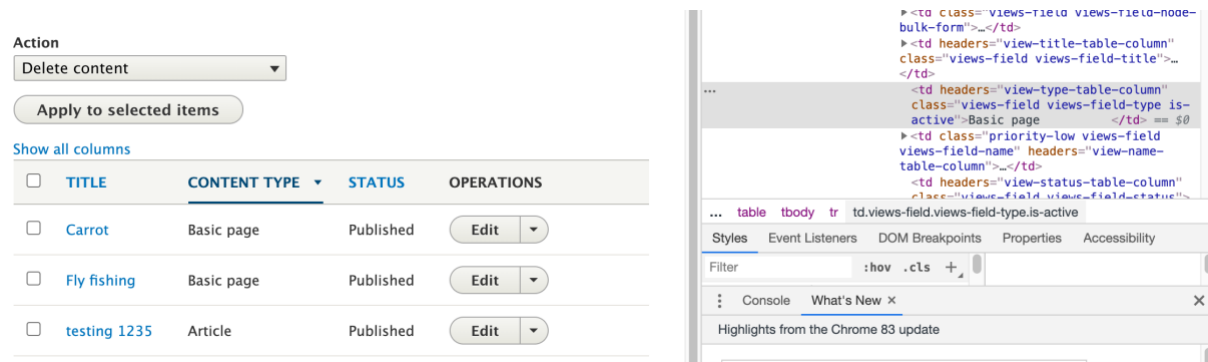


Figure 24: Drupal, view module and code written in different colours

Feature to be tested

The tables creator will be prototyped and tested to combine different potential issues:

- Design of the table for readability – restricted vs flexible to authors.
- Readymade simple tables vs easy ways for the author to create them.
- Clear instructions for more complex tables vs wizards or similar.

5.5 Forms editor

A forms editor is used to create different kinds of forms that may be used for registration, contact or any other interaction with the user of the website. Forms can be very simple (fill in your email address to subscribe to our newsletter) or extremely complex, with backend logic to save content or present material from other systems, validation or authentication as well as many pages and steps. As the form requires interaction with the user, accessibility is key. At the same time, forms require both output and input to work in a correct way, and there is no surprise that forms are the objects where most accessibility issues are normally found.

For many users, forms can be a challenge. This is true for users with cognitive impairments, motor impairments and users with any kind of assistive technology, whether it is a screen reader, an enlargement tool or an input device.

The most common success criteria fails found in forms are:

- 1.3.5 Identify Input Purpose (Level AA)
Fields often lack clear descriptions about what the visitor should fill in and in which format.
- 1.4.1 Use of Colour (Level A)
Error messages or indications are often marked with colour alone.
- 1.4.10 Reflow (Level AA)
Forms built with multiple columns don't work well in responsive mode.
- 1.4.11 Non-text Contrast (Level AA)
Often low contrast on text explaining error messages.

- 1.4.13 Content on Hover or Focus (Level AA)

Tips are often incorrect when it comes to hover and time limits.

- 2.1.1 Keyboard (Level A)

Keyboard navigation is often incorrect, especially for radio buttons and check boxes, which are to be handled with arrow keys, not tab.

- 2.1.2 No Keyboard Trap (Level A)

Incorrect forms can make life difficult for Assistive Technology users, for example when IDs are not unique.

- 2.4.3 Focus Order (Level A) (ID29)

Placement of buttons are often not consistent; column layout is often the reason for this.

- 2.4.4 Link Purpose (In Context) (Level A) (ID30)

Help is often named "help" only for each field and is not connected to the corresponding fields.

- 2.4.6 Headings and Labels (Level AA) (ID32)

Form labels are missing, especially for radio buttons and check boxes where fieldset should be used.

- 2.4.7 Focus Visible (Level AA) (ID33)

Visual focus is often missing for parts of form objects.

- 2.5.1 Pointer Gestures (Level A) (ID34)

When it is possible for the author to create sliders for example.

- 2.5.2 Pointer Cancellation (Level A) (ID35)

Buttons are sometimes onclick instead of ordinary buttons.

- 2.5.3 Label in Name (Level A)

Especially in order forms with multiple options, the technical description does not start the same way as the visual description.

- 3.2.1 On Focus (Level A)

- 3.2.2 On Input (Level A)

Both of the above are less frequent these days, but still occur.

- 3.2.3 Consistent Navigation (Level AA)

Buttons are presented in a different order in different places (cancel, send etc).

- 3.2.4 Consistent Identification (Level AA)

Naming is not consistent (for example send buttons).

- 3.3.1 Error Identification (Level A)

Error handling is one of the most recurring mistakes, should be text based.

- 3.3.2 Labels or Instructions (Level A)

Placeholder is used without label or other support for assistive technology. Furthermore, the placeholder text may either disappear or decrease in size until it is not readable when the user starts filling in the form field. The visual description should be permanent.

- 3.3.3 Error Suggestion (Level AA)

Error messages are often pre-coded. Authors can't (or won't) change/correct them.

- 3.3.4 Error Prevention (Legal, Financial, Data) (Level AA)

Summary often lacking before submitting order forms.

- 4.1.1 Parsing (Level A)

Validation errors in forms are abundant.

- 4.1.2 Name, Role, Value (Level A)

Incorrect use or lack of ARIA.

- 4.1.3 Status Messages (Level AA)

ARIA live is not used in error messages in a correct way.

Example: Drupal

Interactive web forms are a huge challenge for many websites and authoring tools. Drupal has been built around a Form API which allows for developers to produce a vast array of standardised HTML input types.¹³ All of the forms for Drupal are managed through the Forms API, so it was easier to build for ATAG 2.0 - Part A, because the functionality that was needed on the back end, was also exposed to the frontend.

[Drupal: Form API](#)

In Drupal 7, the web forms were built to comply with WCAG 2.0 AA. In Drupal 8, more WAI-ARIA support and reviewed elements for WCAG 2.1 AA compliance were added as well. The Drupal community is always working to see that interactive elements are more semantically defined. If the Inline Form Error module is enabled in Drupal 8, then 3.3.1 Error Identification can be satisfied.¹⁴ This applies to all of the forms in both the frontend and backend.

[Drupal: Inline Form Errors module overview](#)

Most authors using Drupal would have limited experience building web forms through the administration interface. Administrators, however, can create content types and feedback forms. The Drupal webform module changes this.¹⁵ It is much more common in Drupal for authors to need to add, edit and review the feedback from webforms produced by this module. This is most commonly used for surveying users. Drupal 8's Webform module may be one of the only one that aspires to aspects of ATAG 2.0 AA.¹⁶ Much of the backend with the Webform module is accessible thanks to Drupal 8's FAPI, but the webform team went further.¹⁷

[Drupal: Webform module](#)

[Drupal: Webform features](#)

[Drupal: Webform accessibility](#)

¹³ <https://www.drupal.org/docs/drupal-apis/form-api>

¹⁴ <https://www.drupal.org/docs/8/core/modules/inline-form-errors/inline-form-errors-module-overview>

¹⁵ <https://www.drupal.org/project/webform>

¹⁶ <https://www.drupal.org/docs/8/modules/webform/webform-features>

¹⁷ <https://www.drupal.org/docs/8/modules/webform/webform-accessibility>

Webform Accessibility

Last updated on 30 January 2019



Drupal and the Webform module strives to be fully accessible to all users and site builders. Assistive technologies, including screen readers and keyboard access, are fully supported.



The Webform module is committed to being accessible to everyone

View our [self-assessment](#) | Learn more [about our commitment](#)

Below is information and resources for testing and reviewing webform accessibility

Drupal Form Accessibility

- [Accessibility | Drupal.org](#)
- [Drupal 8 Accessibility](#)

Figure 25: Drupal, webpage showing webform accessibility information

On this page

- [Drupal Form Accessibility](#)
- [Drupal General Accessibility Resources](#)
- [Form Accessibility Resources](#)
- [General Accessibility Resources](#)
- [Guidelines](#)
- [Examples and Patterns](#)
- [Checklists](#)
- [Recommended Accessibility Evaluation Tools and Extensions](#)
- [Recommended Screen Readers](#)
- [Videos](#)
- [Blog posts](#)
- [Webform Accessibility Issues](#)

Of the items to highlight, the module prominently displays their accessibility self-assessment¹⁸ and actively encourages users to learn more about their commitment to inclusion.¹⁹ They have identified and addressed an impressive number of accessibility bugs in their issue queue.²⁰ With each release of the Webform module, Pa11y,²¹ an automated accessibility testing tool is used to scan example webforms and check for regressions.²²

[Drupal: Webform module for Drupal 8](#)

[Drupal: and their commitments to inclusion](#)

[Drupal: Issues for webforms](#)

[Pa11y: Open source tools for more accessible web pages](#)

[Drupal: Code to check for regressions](#)

The module has a number of sub-modules. One of them includes a number of sample Webforms which have been built to demonstrate accessibility best practices.

¹⁸ <https://docs.google.com/spreadsheets/d/19OJCDet7RF6pXmnSEq1-5EVvQPpU8VFFuD0ADxRtKb4/edit#gid=949844770>

¹⁹ <https://opencollective.com/webform/updates/webform-module-for-drupal-8-diy-accessibility>

²⁰ <https://www.drupal.org/project/issues/search/webform?text=&assigned=&submitted=&project=issue+followers=&version%5B%5D=any+8.x-&issue+tags+op=%3D&issue+tags=Accessibility>

²¹ <https://pa11y.org>

²² <https://git.drupalcode.org/project/webform/-/tree/8.x-5.x/reports/accessibility/text>

- Webform Examples** ▶ Provides examples of all webform elements and functionality which can used for demonstrating and testing ad
- Webform Examples Accessibility** ▶ Provides example webforms for reviewing and testing accessibility.
- Webform Handler Example** ▶ Provides an example of a webform handler.
- Webform Remote Post Example** ▶ Provides an example of a webform submission posted to a remote server.
- Webform Variant Example** ▶ Provides an example of webform variants.

Figure 26: Drupal, showing webform options with checkboxes

After enabling it, you can evaluate the examples provided:

18 webforms

TITLE	DESCRIPTION	CATEGORY	STATUS	AUTHOR	RESULTS	OPERATIONS
Contact	Basic email contact webform.		Open		0	Build ▾
Example: Accessibility Advanced	Advanced webform accessibility example.	Example: Accessibility	Open		0	Build ▾
Example: Accessibility Basic	Basic webform accessibility example.	Example: Accessibility	Open		0	Build ▾
Example: Accessibility Containers	Container webform accessibility example.	Example: Accessibility	Open		0	Build ▾
Example: Accessibility Labels & Descriptions	Example of webform label and description accessibility.	Example: Accessibility	Open		0	Build ▾
Example: Accessibility Wizard	Wizard webform accessibility example.	Example: Accessibility	Open		0	Build ▾
Example: Computed	Examples of a computed elements.	Example	Open		0	Build ▾

Figure 27: Drupal, webform options visual in different tables and categories

An author should be able to “see” the accessibility and follow best practices. Exposing them is useful for those that want to do the right thing.

Example: Accessibility Wizard

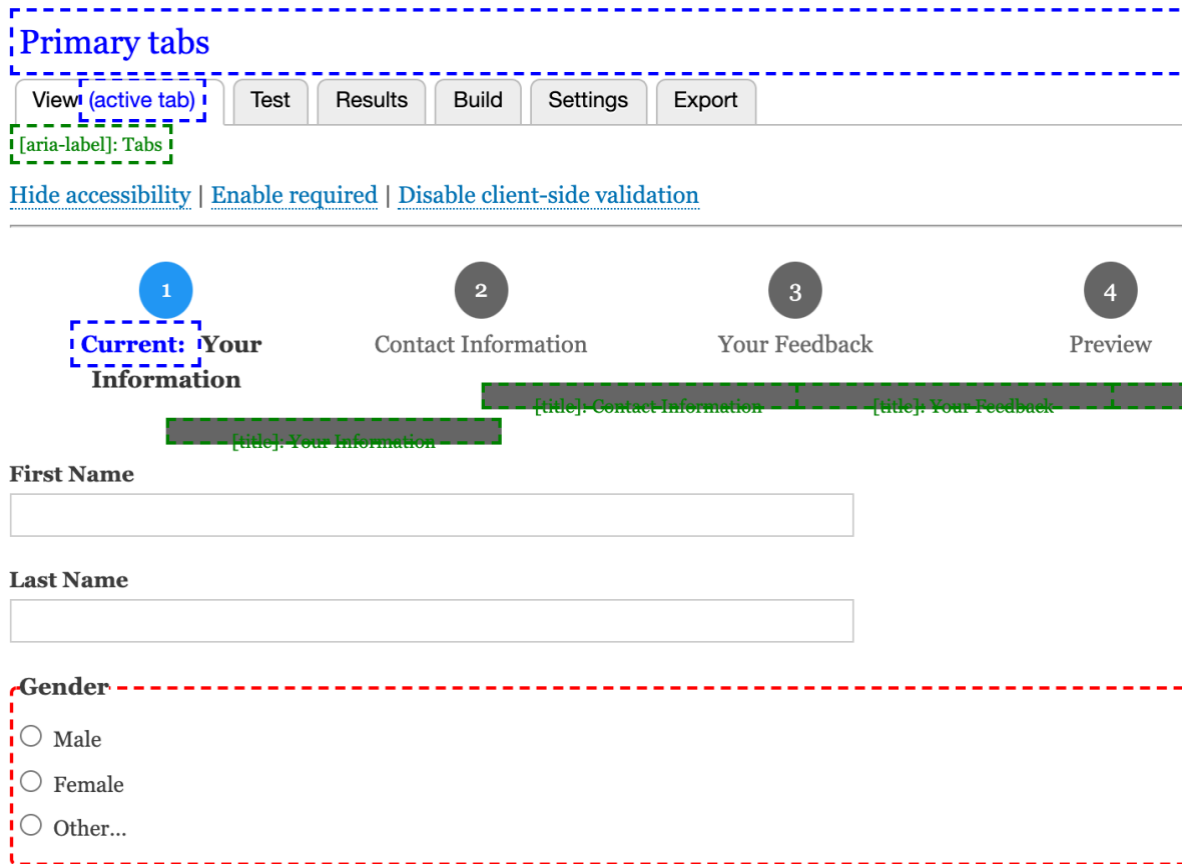


Figure 28: Editor view showing contact information boxes to fill out

Authors are given the option to “Show accessibility”, “Enable required” and “Disable client-side validation” for accessibility testing. Providing a range of forms to provide testing is quite useful.

They also include customisable required messages:

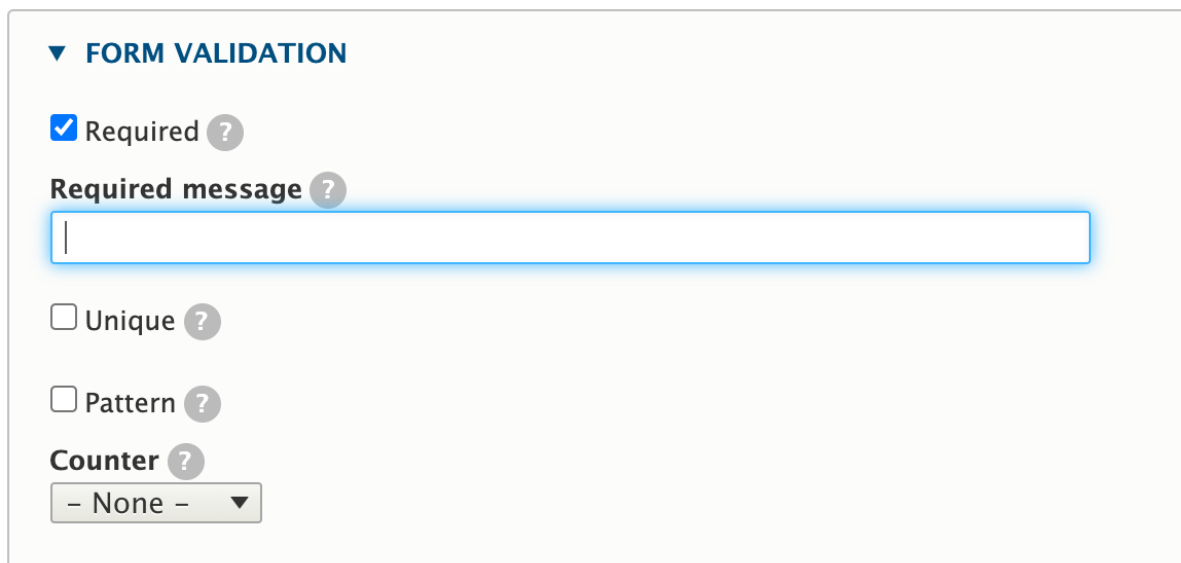


Figure 29: Form validation boxes showing required message, with clickable options

▼ **FORM VALIDATION**

Required ?

Required message ?

Unique ?

Pattern ?

Counter ?

- None - ▼

Required message

If set, this message will be used when a required webform element is empty, instead of the default "Field x is required." message.

Figure 30: Form validation, required message question mark option, shown in yellow

The help text is useful as well to explain how it replaces the default provided by Drupal.

Example: Plone

Plone 6 / Volto comes with a new forms builder that allows editors to create forms without the need to know or understand HTML forms. Since the forms are auto generated, we can make sure that the generated HTML code follows accessibility best practices.

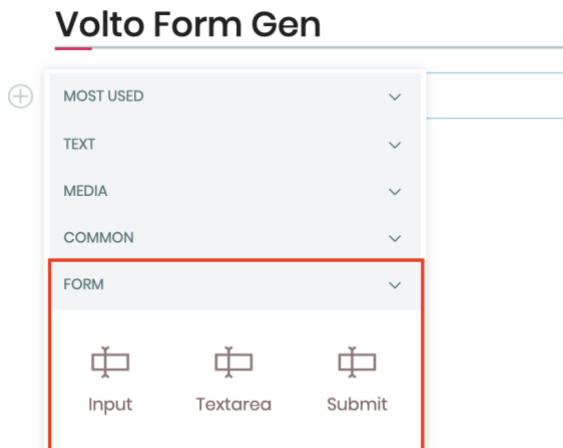


Figure 31: Volto editor, drop down menu showing different form options

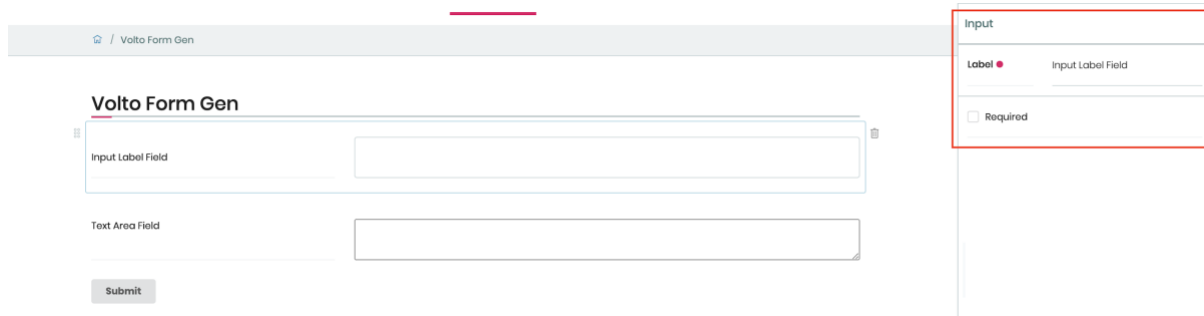


Figure 32: Volto editor, form showing different boxes to be filled out

Features to be tested

The forms creator will be prototyped and tested to focus on basic forms:

- Supporting assistive technology (labels, keyboard navigation etc).
- Consistency and design.
- Avoid mistakes.
- Understandable error messages.

5.6 Video

Video is a very important format to provide accessible content. According to OECD research (IALS²³), as many as 25% of the adult population in most EU countries have difficulties reading and understanding a news article, and YouTube is competing with Google as the preferred search engine.

For users with hearing impairments, captioning is essential. For users with visual impairments, audio description is important. As live video is exempt from the Web Accessibility Directive, the features tested in the project will focus on pre-recorded videos.

Most authoring tools come with some kind of possibility to publish videos, either locally hosted videos or by embedding external video services. The most basic need when embedding videos is to be able to ensure that the embedding is done in an accessible way. That can be as easy as making sure that the iframe that is normally created contains a description of the video content by adding the title attribute in the iframe element.

When adding a locally hosted video, it's important that the author has the possibility to add captions and audio descriptions.

Example: Plone

Plone comes with a Video block (YouTube, Vimeo, MP4) without accessibility support.

²³ (OECD 2000, IALS)

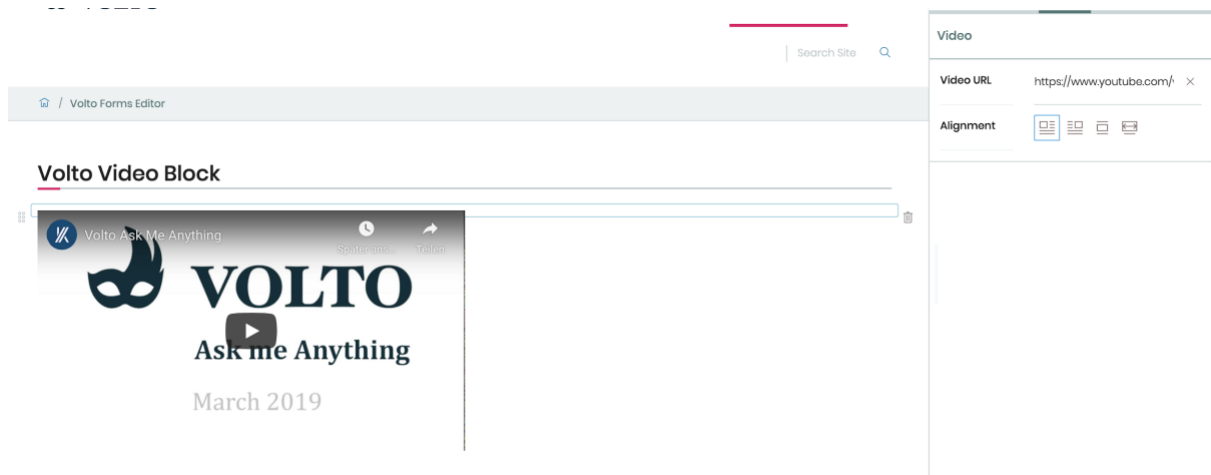


Figure 33: Volto editor, showing video player and URL input box.

Example: Drupal

Drupal's media library does not have this functionality yet, but it is an issue under consideration for Drupal 9.1.²⁴ It is possible in Drupal 8 to configure the Media module to allow fields for transcripts to be uploaded.

[Drupal: Tools for transcripts and captions is under consideration for Drupal 9.1](#)

Features to be tested

The video feature will be tested when it comes to embedded as well as locally hosted videos, including different options for adding captioning and audio descriptions.

5.7 Live testing while authoring

To validate your work while writing is something many authors are used to and appreciate when it comes to spellchecking and similar functionalities. In the same way, accessibility mistakes can be pointed out and corrected immediately. This way, inaccessible content can be avoided and the burden of making accessibility tests and remediations tend to feel less cumbersome.

A live accessibility check could be done in the WYSIWYG editor while publishing, inspecting the content and supporting the author with help texts and easy to understand suggestions for remediation.

Example: TinyMCE accessibility checker plugin

²⁴ <https://www.drupal.org/project/drupal/issues/3002770>

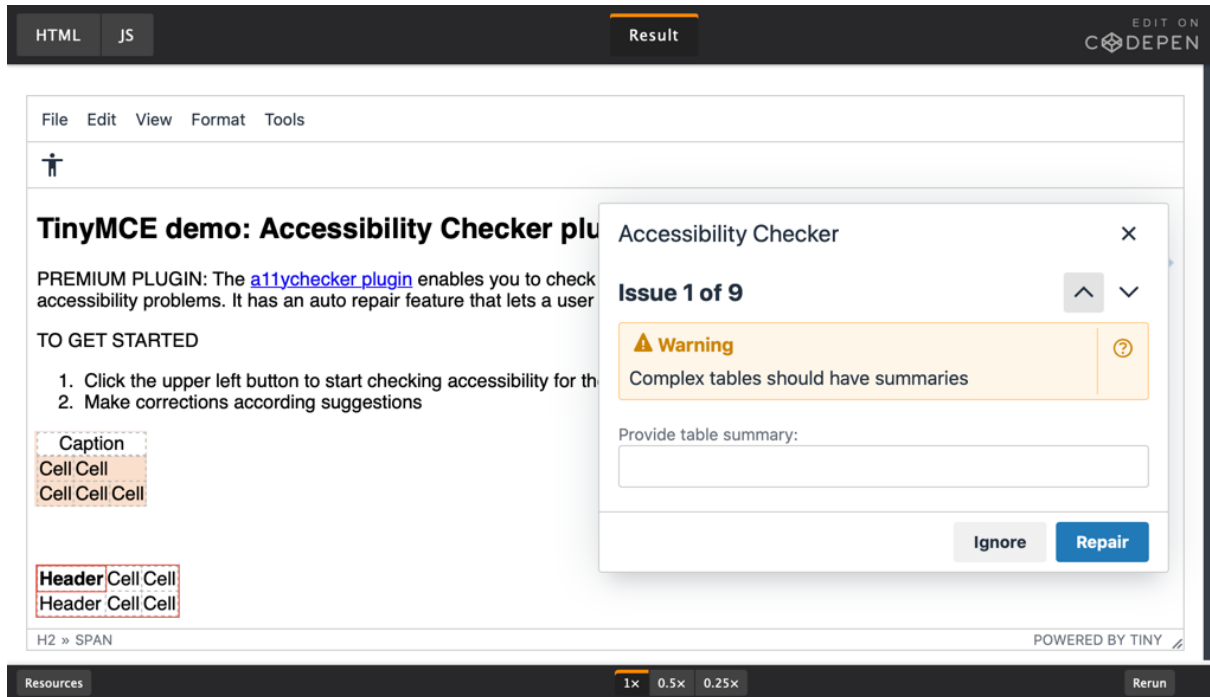


Figure 34: TinyMCE, editor showing accessibility checker box with warning message

When you run the commercial accessibility check, you get a warning with a short description and a link to more in-depth information around that specific rule.

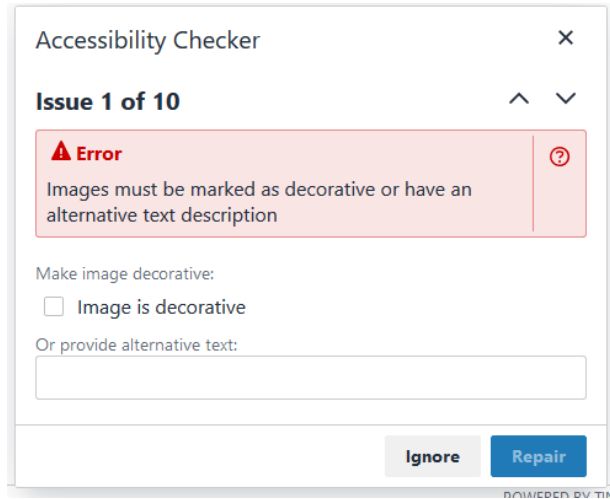


Figure 35: TinyMCE, editor showing error message in accessibility checker

The question mark leads you to the TinyMCE website that has descriptions of the ruleset that is checked.²⁵ The Accessibility Checker aims to be WCAG2.1 AA compliant.

[TinyMCE: Accessibility checker plug in](#)

There is an online demonstration of the accessibility checker in TinyMCE:²⁶

²⁵ <https://www.tiny.cloud/docs/plugins/a11ychecker/>

²⁶ <http://fiddle.tinymce.com/glhaab>

Features to be tested

Live testing while authoring will be tested when it comes to presentation of errors and feed back to the author. This has to be clear but at the same time not too disturbing.

5.8 Testing of documents

Inaccessible documents may be one the most frequent accessibility issue across public sector websites in the EU. It is also an issue most web authors claim to be the most urgent to solve.

Documents can be just as accessible or inaccessible as html. While most page templates, objects and functions of the website are usually made by the authoring tool framework or the supplier of the website, the basic technical parts of accessibility, such as code validation etc., can be accessible provided that the authoring tool provider and the IT supplier know what they are doing. In this case, the web author can still “break” accessibility while publishing, but at least basic accessibility is built in from the start.

For documents, on the other hand, there are usually many different kinds and a lot of variation. Some of the documents may be produced using a template, but many – rarely the majority – are not. This means that basic technical accessibility problems occur over and over again risking, for example, the exclusion of screen reader users from the content of the documents.

When it comes to readability, design and UX related issues, documents are more similar to html in the sense that the level of accessibility has to do with knowledge.

Documents can exist in different formats, but the most common format when discussing inaccessible documents in public sector websites is PDF. These PDFs can essentially stem from three types of sources:

- Web authors create a Word document and convert it to PDF.
- External suppliers provide designed brochures and similar items as PDF documents.
- Internal systems automatically generate PDF documents.

When web authors create documents, the lack of accessibility often has to do with lack of knowledge, or sometimes lack of time. This can in theory be resolved with training and manuals, but in reality, inaccessible PDFs created by web authors are abundant in public sector websites in the EU.

When external suppliers deliver documents, the lack of accessibility usually has to do with the lack of requirements made by the website owner. This is often easy to fix provided the procuring party is aware of it, as the supplier can use a third party to train them or remediate the documents.

When internal systems generate documents, the lack of accessibility may be harder to mitigate, as complex systems can be resource demanding to change.

As many web authors feel insecure about how to make their documents accessible (or they may not even know they should do it), a document testing feature would be valuable to test the accessibility of the document before publishing it.

Automatic testing of document accessibility already exists on the market, both as external commercial products and built into the PDF making software Adobe Acrobat DC. As with any automatic accessibility testing, they are not able to find all accessibility issues, but the automated tests do cover the key issues that may for example exclude screen reader users from the content.

The Adobe accessibility checker could potentially be included in the authoring tool by creating an API based service that can be used to test the document before publishing it. This way, the author would be made aware about issues that can be automatically tested, such as tagging of the document, which is important for assistive technology to navigate the content. The automatic checker can also point to potential issues that have to be corrected manually, like colour contrast and reading order.

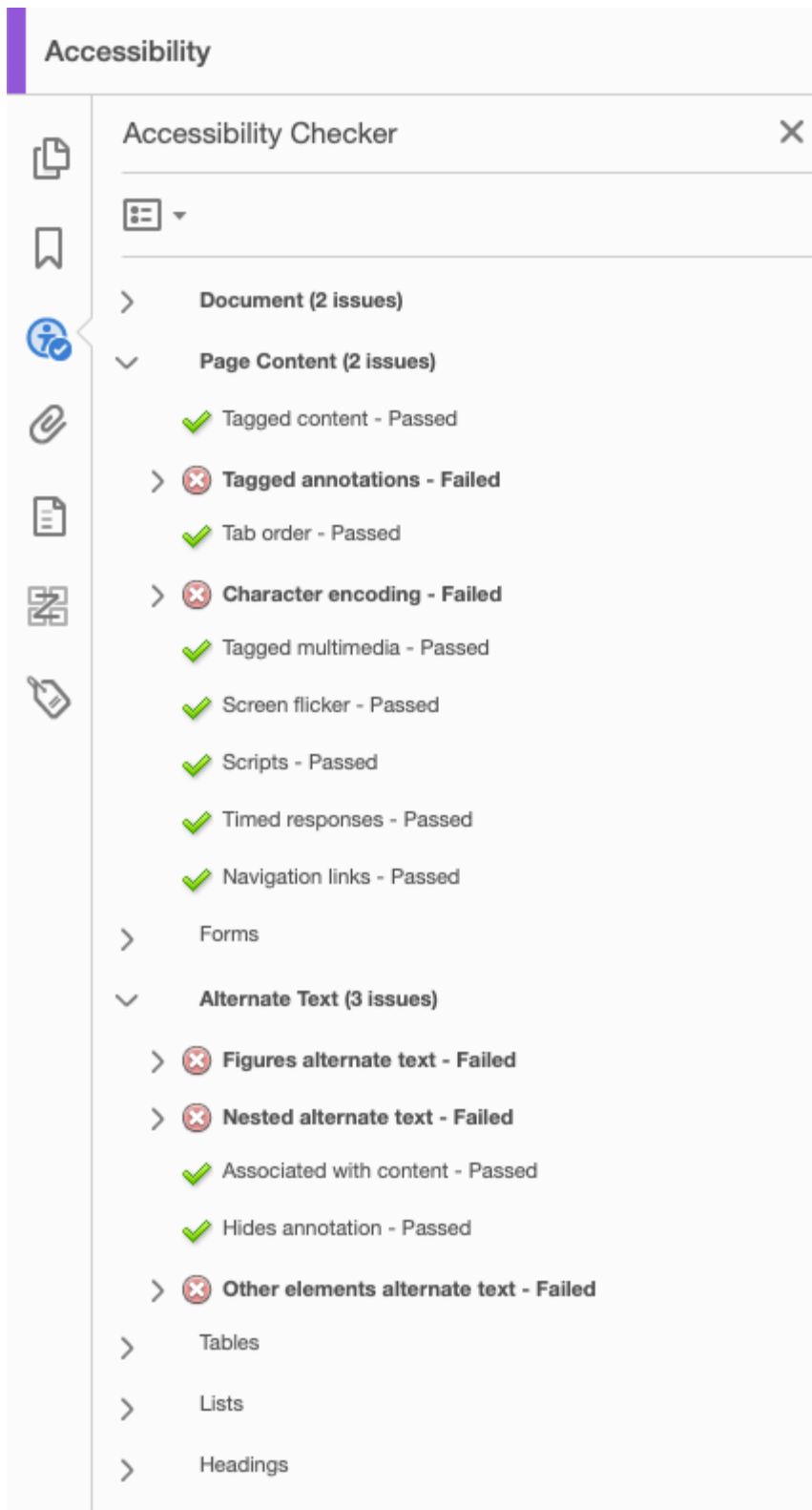


Figure 36: Acrobat pro accessibility checker showing checkboxes, fails and pass

Features to be tested

- At what point in the publishing process is this test most efficient.
- How are the errors to be reported in the best way to the author.

5.9 Testing the content of pages

Most people who create content on the websites of public sector authorities in the EU are not trained communication professionals and the vast majority are not trained in accessibility (or even aware of the requirements). Therefore, an accessibility check before publishing content would be very useful. In many authoring tools, it is also possible to assign different levels of publishing rights to different authors, so that people who use the tool less often get their content checked and approved by a central communication staff member before publishing. While this is sometimes required for legal or quality reasons, this kind of manual quality assessment (identifying and mitigating potential accessibility flaws) is extremely resource demanding and inefficient, as content is constantly updated.

A built-in accessibility page checker can be very valuable for web authors, to ensure inaccessible content is not unconsciously published, but also to raise awareness and increase knowledge. If the web author repeatedly gets the same complaints from the checker, chances are that they will try to learn how to do it right from the start, rather than remediate afterwards. This way, the built-in automatic checker can have an impact on two levels: in the short term by improving the immediate accessibility of the web page in question and in the longer term by improving general awareness and knowledge about accessibility of the author, a knowledge that can be reused in and spread to other parts of the organisation and beyond.

There are many automatic accessibility testing tools on the market; open source as well as commercial. They vary in quality and complexity, but more importantly, they focus on different target audiences. Many of the tools are made specifically for developers, to support their work while creating a website. Others help designers to choose the right combination of colours etc. Some tools are made to support web authors to publish in an accessible way.

There are also many other kinds of tools that may help the authors, like spellcheckers and crawlers that look for broken links etc.

For a built-in accessibility check to be of real value for the average web author, it is important that the results do not require programming skills. It is also desirable that the result is clear and easy to understand, and solutions are offered. They should not only identify problems.

As with all automatic accessibility testing, these tools can only test parts of the requirements. They also usually miss accessibility problems that are possible to detect automatically, and they also point to problems that are in reality not accessibility issues. Some of them can point to issues that need to be checked manually. But even with these limitations, an automatic accessibility check is still a good start, as it can find basic issues that are easy to remediate - a kind of hygiene factor.

By building an API based service using an automatic accessibility testing tool core service, the page specific content can be checked. This way, only web author relevant failures would be presented, so that the author can concentrate on remediating these.

The main goal of this feature is to help the web author ensure that different objects or parts of the web page are checked separately and that the end result is accessible - before publishing the page.

Example: Plone

Plone 5 comes with an add-on product to integrate Siteimprove accessibility checks:²⁷

[Plone 5 add-on that provides integration with Siteimproves](#)

Example: Drupal

There is an open issue to provide an accessibility checker from within the WYSIWYG²⁸ but currently the upstream CKEditor plugin uses a deprecated accessibility engine which could provide some accessibility problems in the future. In the interim, it is recommended to use the CKEditor accessibility auditor²⁹ module which leverages the popular HTML CodeSniffer engine.³⁰

[Drupal: Open dialog about include Axe accessibility tester in CKEditor](#)

[Drupal: CKEditor accessibility auditor](#)

[Github: HTML Codesniffer engine](#)

Features to be tested

- Presentation of accessibility errors; in context or as a separate list.
- The level of support needed to actually help the authors remediate the errors.
- Possible integration of support.

5.10 Testing the whole website

An accessibility check of the whole website is valuable to the website owner for compliance reasons, but also to be able to detect any specific departments or type of content that needs refinement or groups of staff that need additional training.

As with all automatic accessibility testing, these tools can only test parts of the requirements, see 5.9. As with the page testing feature, the results should be clear, and solutions provided (e.g. links to where errors are found etc.) to make remediation easy.

Some authoring tools use open source accessibility checkers for this kind of tests. The results are mixed, but in general the reports are more aimed for developers than website owners, and they are often so complex that an accessibility expert is needed to interpret and remediate. Providing accessibility testing of the whole website that results in reports that are easier to understand and use would be valuable.

Some commonly used open source checkers:

[Github: Accessibility monitor for open source accessibility checkers](#)

[Github: Axe-crawler for open source accessibility checkers](#)

[Github: AXErunters for open source accessibility checkers](#)

<https://github.com/IBMa/equal-access/>

²⁷ <https://pypi.org/project/collective.siteimprove/>

²⁸ <https://www.drupal.org/project/drupal/issues/2731373#comment-13704500>

²⁹ https://www.drupal.org/project/ckeditor_accessibility_auditor

³⁰ https://github.com/squizlabs/HTML_CodeSniffer

<https://github.com/pa11y>

<https://github.com/Siteimprove/alfa>

Example SiteVision:

SiteVision, has a built an accessibility checker that can generate a report based on the W3C validation service, which identifies basic validation errors. The test can be run on demand or at a regular time set up by the website owner.

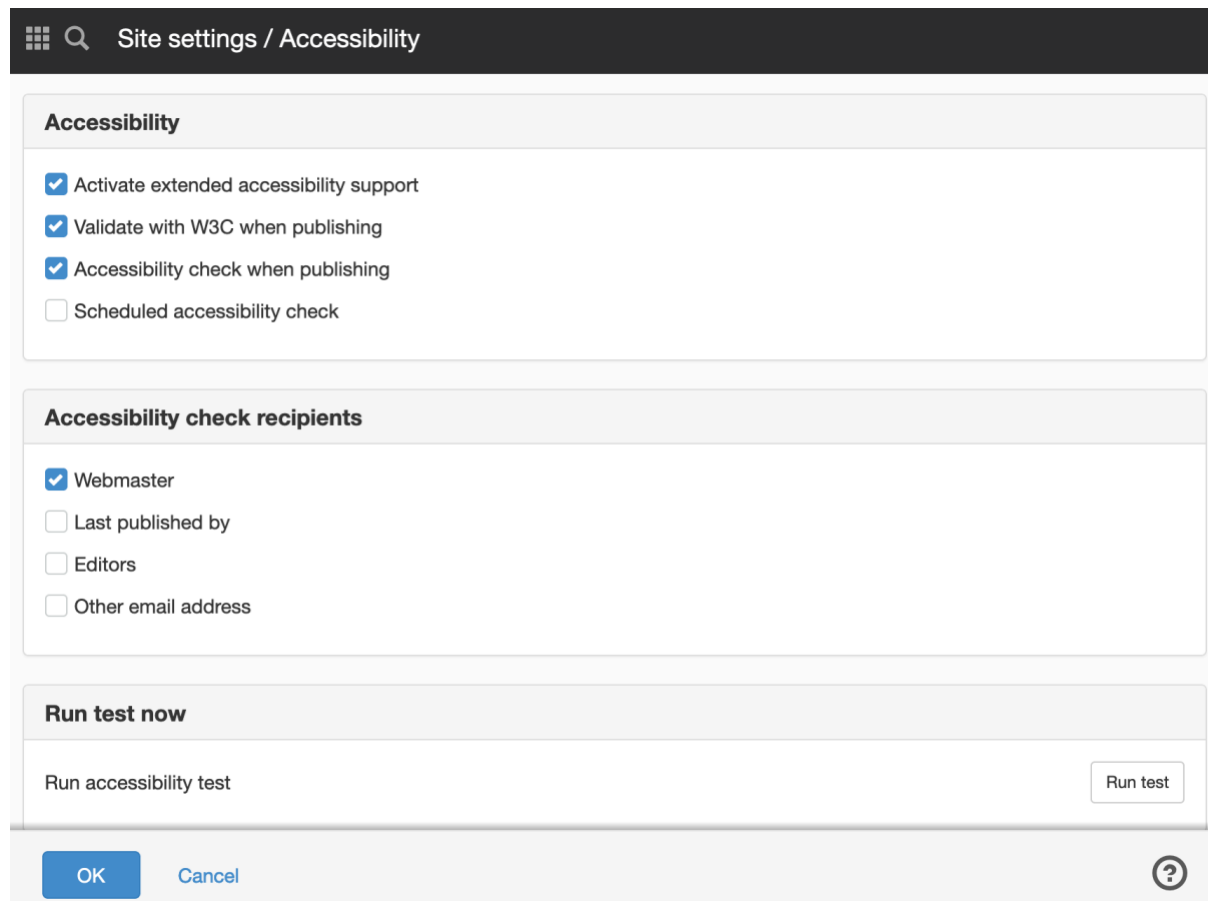


Figure 37: Sitevision, editor showing site settings with checkboxes of accessibility



Figure 38: Sitevision, email showing accessibility test information

Features to be tested

- Error reporting (email, online ...)

- Format of reporting (html, PDF ...)
- Level of detail vs overview in summary

6 Conclusion

Within the consortium and beyond, built-in accessibility features in web authoring tools exist, but there is no common ground as to which features are standard, how they work or who they target. In general, many of the features are leaning heavily towards the developer perspective even when addressing authoring needs.

Building on the experience of the partners and senior advisors, the ideas that have emerged when analysing the user requirements as well as the state of the art, a set of by default accessibility features with great potential have been selected for prototyping and testing.