CROP MARKS · BLEED 3MM · A4 PORTRAIT

WeasyPrint

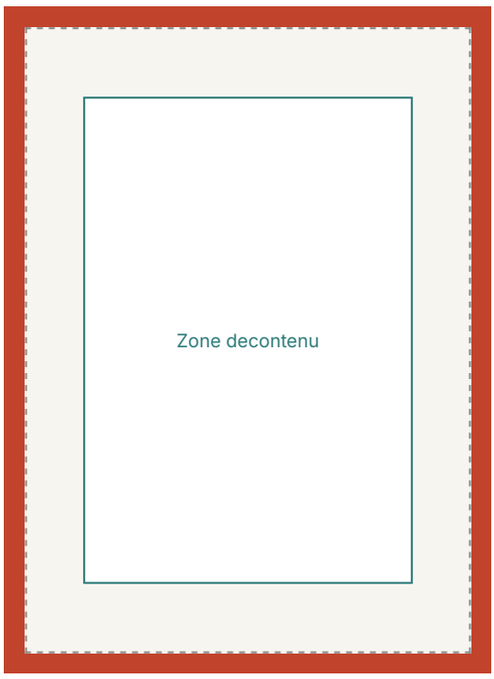
Cheat sheet et limites vérifiées : carnet de tests CSS Paged Media, chaque affirmation prouvée par un build réel.

WEASYPRINT 70.0 · WEASYPRINT.ORG · RAPPORT DE TESTS

Sommaire

Vocabulaire de base

Quatre mots reviennent tout le temps dans ce document. Voici ce qu'ils veulent dire, avec un schéma d'une page vue en coupe.



Page A4

Le format final après impression et découpe : 210 × 297mm. C'est la taille que le lecteur tiendra dans les mains.

Marge

L'espace vide entre le bord de la page et le texte/les images. Empêche le contenu de coller au bord une fois la page découpée.

Bleed (fond perdu)

Une petite bande supplémentaire (souvent 3mm) ajoutée tout autour de la page, au-delà du format final. Sert à éviter un fin liseret blanc si la découpe de l'imprimeur n'est pas parfaite au millimètre près : une couleur ou une image qui doit toucher le bord de la page est étendue jusque dans cette zone.

Marks (repères de coupe)

De petits traits noirs dessinés aux coins de la page imprimée, en dehors du bleed. Ils indiquent à l'imprimeur exactement où couper le papier pour obtenir le format final.

Cheat sheet WeasyPrint

Aide-mémoire des réglages CSS Paged Media à connaître pour produire un PDF imprimable correct avec WeasyPrint : chaque point a été vérifié par un build réel (`scripts/build.sh showcase-magazine.html`) sur WeasyPrint 70.0, pas seulement documenté sur la foi de la spécification CSS.

Le bloc de départ pour une page imprimable

```
@page {
  size: A4 portrait;
  margin: 20mm 15mm;
  bleed: 3mm;
  marks: crop;
}
```

`size` fixe le format final, `margin` l'espace autour du contenu, `bleed` le fond perdu, `marks` les repères de coupe. Voir la page "Vocabulaire de base" pour le détail de chaque terme.

`bleed` ne sert à rien sans `marks`

Déclarer `bleed: 3mm` seul, sans `marks` (même juste `marks: crop`), fait que le bleed est ignoré silencieusement : aucune erreur, mais aucun débord non plus. Toujours mettre les deux ensemble.

Sauts de page : `break-after: page`

Sur un élément englobant une section (une `<article>` par page, par exemple), force le passage à la page suivante. Utilisé sur chaque page de ce document.

Couleurs : `@color-profile` + `--output-intent` pour un vrai CMJN natif

```
@color-profile --fogra39 {
  src: url(chemin/vers/fogra39.icc);
  components: cyan, magenta, yellow, black;
}
```

Puis `weasyprint fichier.html sortie.pdf --output-intent="--fogra39"` . Le nom doit commencer par deux tirets (identifiant CSS personnalisé), et correspondre exactement entre la règle CSS et l'argument CLI. Passer un chemin ICC brut en CLI sans cette règle CSS n'est pas la syntaxe officielle : ça ne fonctionne que par effet de bord. Cette méthode écrit réellement les couleurs en CMJN dans le flux PDF, contrairement à un moteur basé sur un navigateur qui reste en RGB. Limite connue : le groupe de transparence CSS reste force en RGB même dans ce mode.

`device-cmyk()` et son équivalent hex ne donnent pas la même teinte

Écrire directement `device-cmyk(65% 0% 10% 40%)` plutôt que `#2b7a78` produit une couleur légèrement différente une fois convertie en RGB (mesure : 58,129,148 contre 43,122,120). Le gamut CMJN étant plus restreint, les deux ne se valent pas forcément à l'œil. Détails page suivante.

Cheat sheet WeasyPrint (suite)

Fond pleine page (avec bleed) : via `@page`

```
@page cover { background-color: #3d5a80; }
.cover { page: cover; }
```

Peint nativement toute la boîte de page, bleed compris. Vérifie page suivante.

La propriété `filter` (`saturate`, `contrast`, `sepia`...) n'est pas supportée

WeasyPrint utilise Pango, un moteur de mise en page de documents, pas un moteur de rendu web complet comme Chromium : il ignore silencieusement `filter` (warning au build, mais le PDF se génère quand même sans l'effet). Pour appliquer un effet visuel à une image, le faire en amont sur le fichier lui-même (ImageMagick, Affinity), pas via CSS. `opacity` reste supporté normalement, ce n'est pas concerné par cette limite.

Images bitmap : à vérifier au cas par cas après `--output-intent`

`--output-intent` agit sur les couleurs CSS (texte, aplats, bordures), pas forcément sur les pixels d'un JPEG/PNG déjà en RGB. Script dédié disponible si besoin d'un contrôle total : `scripts/convert-cmyk.sh` (ImageMagick + profil FOGRA39 / ISO Coated v2) en amont, sur des fichiers séparés dédiés à la livraison prépresse.

Rendu réel : fond via @page

Vérifier qu'un fond posé via `@page` (plutôt que sur un élément HTML) couvre bien toute la boîte de page jusqu'au bleed, sans bande blanche résiduelle. Cette page utilise `@page bgPage { background-color: #3d5a80; }` : le résultat ci-dessous est la preuve en situation réelle, pas une capture d'écran retouchée.

Mise en page 2 colonnes

`column-count` répartit un bloc de texte sur plusieurs colonnes sans avoir à le couper à la main. La preuve : ce code est lui-même affiché dans les deux colonnes qu'il crée.

```
.article-corps {
  column-count: 2;
  column-gap: 8mm;
  column-rule: 1px solid #ddd;
  text-align: justify;
}
```

`column-count: 2` demande deux colonnes. Le navigateur (ou WeasyPrint au build) calcule seul où couper le texte pour équilibrer les deux, colonne par colonne, sans balise supplémentaire dans le HTML.

`column-gap` fixe l'espace entre les colonnes, ici 8mm. `column-rule` ajoute le filet vertical visible entre elles, avec la même syntaxe qu'une bordure CSS classique (épaisseur, style, couleur).

Si le contenu est trop court pour remplir les deux colonnes, la seconde reste vide en bas plutôt que de forcer un équilibrage force. C'est le cas ici : ce paragraphe termine le texte.

Image RGB vs CMJN : même photo, deux fichiers

Montrer concrètement la conséquence d'un point déjà établi : WeasyPrint ne convertit jamais les images bitmap (JPEG/PNG) en CMJN au build, seuls le texte et les aplats CSS peuvent l'être via `--output-intent`. Pour une vraie image CMJN, il faut donc un fichier CMJN séparé, produit en amont par `scripts/convert-cmyk.sh` (ImageMagick + profil FOGRA39) : les deux versions ci-dessous en sont la preuve, fichier par fichier.

Original

Fichier `jamo-images-f2CPkHubfLc-unsplash.jpg`, colorspace sRGB (vérifié avec `identify`).

Convertie

Fichier `jamo-images-f2CPkHubfLc-unsplash-cmyk.jpg`, colorspace CMYK (vérifié avec `identify`), généré par le script.

À l'écran, les deux versions se ressemblent fortement (le navigateur reconvertit toujours en RGB pour l'affichage). La vraie différence est dans le fichier lui- même, pas dans ce qui est visible ici : seule la version de droite contient des données CMJN natives, exploitables telles quelles par une chaîne d'impression professionnelle.

device-cmyk() vs hex : une vraie différence de teinte

Écrire une couleur en `device-cmyk()` ou en `#hex` avec la même intention visuelle ne produit pas la même teinte à l'écran. Trois exemples ci-dessous, chaque paire visant la même couleur :

ÉCRIT EN `device-cmyk()`

`device-cmyk(65% 0% 10% 40%)`

Mesuré à l'écran : RGB 58, 129, 148.

ÉCRIT EN `#hex`

`#2b7a78`

Mesuré à l'écran : RGB 43, 122, 120. **Écart moyen.**

ÉCRIT EN `device-cmyk()`

`device-cmyk(0% 85% 75% 5%)`

Mesuré à l'écran : RGB 229, 52, 59.

ÉCRIT EN `#hex`

`#c1432c`

Mesuré à l'écran : RGB 193, 67, 44. **Écart le plus fort.**

ÉCRIT EN `device-cmyk()`

`device-cmyk(55% 0% 60% 35%)`

Mesuré à l'écran : RGB 79, 140, 86.

ÉCRIT EN `#hex`

`#588157`

Mesuré à l'écran : RGB 88, 129, 87. **Écart le plus faible.**

La différence vient du gamut CMJN, plus restreint que le RGB : une conversion CMJN vers RGB ne retombe jamais exactement sur la même teinte que le code hex choisi à l'œil. L'écart n'est pas une constante fixe : il dépend de la couleur de départ et de sa position dans le gamut (moins de 15 points par canal sur le vert, plus de 35 sur le rouge). En pratique : préférer un code hex choisi à l'œil sur écran, et ne passer par `device-cmyk()` que si une valeur CMJN précise est imposée par un cahier des charges d'impression.

Détail technique : les valeurs RGB ci-dessus viennent d'un build **sans** `--output-intent` (mesure pixel par pixel avec `pdftoppm`), donc de la formule de conversion CMJN vers RGB de WeasyPrint pour son rendu par défaut. Avec `--output-intent` actif (page précédente), le PDF écrit les 4 valeurs CMJN d'origine telles quelles, sans conversion (vérifié avec `mutool show`) ; c'est alors le logiciel qui affiche le PDF (Affinity, via son profil ICC) qui produit une troisième teinte à l'écran, encore différente des deux ci-dessus. Nuancier multi-profils complet page suivante.

Nuancier multi-profils : FOGRA39 vs les autres

Vérifier qu'un même RVB ne se convertit pas en la même recette d'encre selon le profil ICC visé : cinq profils CMJN sont comparés ici sur les mêmes couleurs, pour objectiver ce que change réellement le choix d'un profil plutôt qu'un autre. Teintes de test choisies pour couvrir des cas difficiles en CMJN (rouge et magenta saturés, bleu vif, noir riche), sans reproduire de logo de marque. Chaque profil est converti depuis le même RVB source via `PIL.ImageCms` (intent Perceptual + compensation du point noir). Seul FOGRA39 (colonne grisée) est utilisé en production par `scripts/svg-to-cmyk.sh`, cohérent avec le profil embarqué dans le PDF via `--output-intent`. Les autres colonnes sont conservées ici à titre de documentation/référence.

Couleur	RVB original	**FOGRA39** coated, prod.	**FOGRA27** coated	**FOGRA40** journal SC	**FOGRA47** non couché	**SWOP** Artifex, US
Rouge	 `rgb(227,6,19)`	 `0/96/92/4`	 `0/95/95/4`	 `0/94/98/4`	 `0/95/94/0`	 `4/100/100/0`
Bleu	 `rgb(0,87,184)`	 `93/64/0/0`	 `89/67/0/0`	 `95/70/0/2`	 `98/71/0/1`	 `92/71/0/0`
Orange	 `rgb(255,130,0)`	 `0/58/94/1`	 `0/58/94/1`	 `0/55/100/2`	 `0/52/100/1`	 `0/61/100/0`
Vert	 `rgb(0,166,81)`	 `80/0/87/3`	 `77/0/82/3`	 `83/0/92/2`	 `81/0/98/1`	 `82/7/96/0`
Magenta	 `rgb(230,0,126)`	 `1/96/5/2`	 `1/95/5/3`	 `1/98/15/3`	 `1/100/0/4`	 `3/100/13/0`
Noir riche	 `rgb(26,26,26)`	 `76/68/59/87`	 `68/59/49/87`	 `85/80/82/91`	 `93/79/35/98`	 `70/67/66/78`
Jaune doré	 `rgb(255,215,0)`	 `3/13/91/1`	 `4/13/90/1`	 `4/9/94/1`	`0/7/99/4`	`1/13/100/0`

Constat général : au sein même de la famille FOGRA, les valeurs varient sensiblement selon le support papier visé (coated vs papier journal vs non couché), en particulier sur le noir riche (K de 87% à 98%) et le rouge saturé. Le profil SWOP (US) donne pour le rouge et le magenta des valeurs Y et M plus extrêmes (proches de 100%) que les profils FOGRA (Europe). Aucun profil n'est « le bon » dans l'absolu : le choix dépend uniquement du papier et de la presse réellement utilisés à l'impression. FOGRA39 (colonne grisée) reste le profil de production de ce projet car c'est celui déclaré dans `--output-intent` au moment du build ; changer de colonne sans changer l'output-intent créerait une incohérence entre la couleur écrite dans le SVG et le profil réellement embarqué dans le PDF.

Transparence (rgba) : une limite invisible à l'écran

Ce test superpose deux couleurs avec de la transparence, pour vérifier si l'effet de transparence lui-même pose problème quand le PDF est construit en CMJN. Première étape : voir les deux couleurs de départ, chacune seule et en pleine opacité (aucune transparence pour l'instant).

Couleur arrière-plan `#c1432c` + couleur avant-plan `rgb(45, 90, 200)`, toutes deux en pleine opacité pour l'instant.

Deuxième étape : le même bleu, mais rendu semi-transparent (50% d'opacité). Deux écritures CSS différentes existent pour obtenir exactement le même résultat, comparées ci-dessous :

Hex + `opacity`

```
background: #2d5ac8;
opacity: 0.5;
```

La couleur ET tout le contenu de l'élément deviennent transparents (texte, bordures inclus).

`rgba()`

```
background: rgba(45, 90, 200, 0.5);
```

Seule la couleur de fond est transparente. Le texte à l'intérieur, s'il y en à, reste net. C'est cette version qui est utilisée pour la suite du test.

Transparence (rgba) : le résultat superposé

Vérifier que la transparence CSS se comporte normalement une fois les deux couleurs superposées, avant de chercher plus loin un éventuel problème lié au CMJN. En posant la couleur avant-plan semi-transparente (`rgba(45, 90, 200, 0.5)`) sur la couleur arrière-plan (`#c1432c`) vue à la page précédente, les deux couleurs se mélangent visuellement : c'est le rendu normal et attendu d'une transparence CSS.

Arrière-plan `#c1432c` + avant-plan `rgba(45, 90, 200, 0.5)` = mélange visible ci-dessus.

La question à vérifier maintenant : est-ce que ce mélange change si on demande à WeasyPrint de construire le PDF en CMJN plutôt qu'en RGB normal ? Deux builds du même fichier HTML, avec une seule option en plus sur le second :

Build normal

`weasyprint showcase-magazine.html showcase-magazine.pdf` (sans --output-intent)

Build en CMJN

`scripts/build.sh showcase-magazine.html` (--output-intent="--fogra39" systématique, profil déclaré via `@color-profile` en CSS)

Réponse : rendu strictement identique. À l'œil, rien ne permet de distinguer la version RGB de la version CMJN.

Transparence (rgba) : où se cache la vraie différence

Puisque les deux PDF de la page précédente se ressemblent à l'œil, la seule façon de savoir si le second est vraiment en CMJN est d'ouvrir le fichier avec un outil technique d'inspection PDF (ici `mutool`), qui montre le code interne invisible à l'affichage normal. Voici ce que cet outil affiche pour la zone de la couleur avant-plan transparente, dans chacun des deux fichiers :

Ce qu'on attendrait (tout en CMJN)

```
/Group <<
  /Type /Group
  /S /Transparency
  /CS /DeviceCMYK
>>
```

Le fichier déclare cette zone en CMJN, comme le reste du document.

Ce qu'on trouve réellement

```
/Group <<
  /Type /Group
  /S /Transparency
  /CS /DeviceRGB
>>
```

Même dans le fichier construit avec `--output-intent`, cette zone précise reste déclarée en RGB.

Autrement dit : demander un export CMJN convertit bien la plupart du document, mais oublie chaque zone qui utilise une transparence CSS. C'est un défaut connu et documenté de WeasyPrint (ticket GitHub #2723), pas une erreur dans la configuration utilisée ici.

Consequence concrete : un imprimeur professionnel très strict sur la norme PDF/X-4 (qui exige un fichier 100% CMJN, sans aucune exception) peut refuser ce fichier à cause de cette seule zone RGB résiduelle, même si elle s'affiche correctement partout ailleurs. Pour un usage d'entraînement personnel avec une finition faite ensuite dans Affinity, ce n'est pas bloquant : Affinity reconvertira le fichier de toute façon au moment de l'export final.

Rendu réel : `filter` CSS non supporté

Le CSS ci-dessous applique un filtre visuel à l'image de gauche. Si les deux images se ressemblent, c'est la preuve que WeasyPrint (moteur Pango) ignore `filter`, contrairement à un navigateur complet comme Chromium. Le build affiche d'ailleurs un avertissement à ce sujet : `WARNING: Ignored `filter: saturate(0.4) contrast(1.3) sepia(0.2)`, unknown property.`

filter appliqué en CSS

`filter: saturate(0.4) contrast(1.3) sepia(0.2)` (code présent, effet ignoré au rendu)

Image sans filtre (référence)

Même image, aucun filtre appliqué.

Pour un vrai effet visuel sur une image destinée à WeasyPrint, appliquer le traitement en amont sur le fichier lui-même (ImageMagick, Affinity), pas via CSS.