



# Plongée au cœur des PDF

Bonjour !

Je m'appelle Frédéric BISSON, je suis développeur et je travaille à Rouen (Normandie).

On va faire une petite plongée au cœur des PDF pour voir comment ces monstres sont constitués afin de les optimiser.

# SOMMAIRE

- Pourquoi s'intéresser aux PDF?
- D'où vient le format PDF?
- Comment est construit un PDF?
  - Format de données générique
  - Langage graphique
  - Flux et filtres
- Lecture d'un PDF
- Optimisation!
  - Chaînes de caractères
  - Supprimer le superflu
  - Optimiser le code graphique
  - Améliorer l'efficacité des filtres
  - Utiliser le meilleur filtre
  - Regrouper pour compresser
  - Cumuler les filtres
- Et DietPDF?

Dans cette présentation, on va d'abord s'intéresser à pourquoi il faut s'intéresser aux PDF en 2025. On va ensuite regarder d'où vient ce format, comment il est construit, comment un PDF se lit, du point de vue d'une programme et, enfin, tous les points sur lesquels on peut jouer pour optimiser un fichier. On terminera par l'évocation de mon side-project DietPDF qui m'a permis d'être devant vous aujourd'hui.

# Pourquoi encore s'intéresser aux PDF ?

L'objectif de cette présentation n'est pas de vendre le PDF comme la technologie à laquelle tout le monde va s'intéresser demain. Je suis suffisamment réaliste et conscient que le web a parfaitement prouvé sa capacité à évoluer et à s'adapter que ce soit en termes de présentation (notamment pour la transition des ordinateurs de bureau aux smartphones en passant par les tablettes) et en termes d'interactivité.

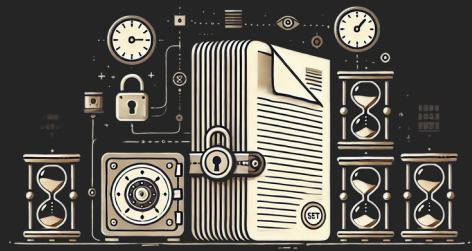
# On n'est pas prêt de s'en débarrasser

- Inconvénients

- Mise en page figée
- Peu d'évolutions
- Peu interactif
- Problèmes d'accessibilité

- Avantages

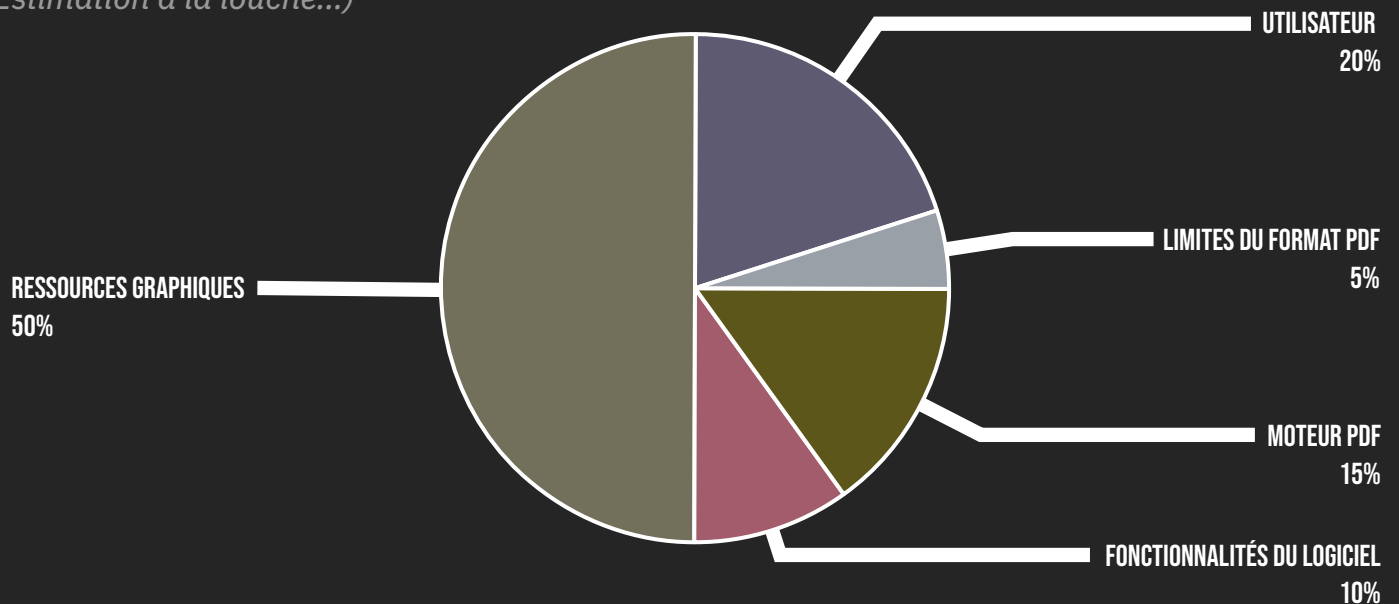
- Structure monolithique
- Adapté à l'archivage
- Signature numérique



Certes, le PDF n'est pas parfait : sa mise en page est figée, le format évolue peu, il est peu interactif et de nombreux problèmes d'accessibilité se font ressentir.

Néanmoins, on n'est pas prêt de s'en débarrasser car il couvre des besoins difficiles à satisfaire avec d'autres technologies. Sa structure monolithique permet le partage et la conservation sans nécessiter de moyens supplémentaires (comme des serveurs, un réseau même local etc.). Il dispose également de la signature numérique.

*(Estimation à la louche...)*



## Qui est responsable du poids d'un PDF ?

Si l'accessibilité d'un PDF se résout avant tout au niveau de l'utilisateur et des ressources graphiques, le poids d'un PDF, lui, est faiblement dû à l'utilisateur. Sur cette estimation à la louche, on peut voir que les principales responsables sont les ressources graphiques incorporées dans le document. Car, oui, les producteurs de PDF ne sont généralement pas des maîtres en matière d'optimisation d'images.

# Problèmes des compresseurs de PDF

- Optimisation ciblée
- Images maltraitées
  - Compression forte
  - Résolution  $\leq 200$  ppp
- Conformité
- Métadonnées
- Side-project : **DietPDF**



Bien sûr, il existe déjà des « compresseurs » de PDF. Vous connaissez sûrement ILovePDF. Adobe propose également ses services... payants. Mais les plateformes qui manipulent et optimisent les PDF ont pour objectif, très souvent, l'affichage écran. Ils vont donc réduire la résolution des images ou en augmenter le taux de compression. Ils suppriment même des données et des métadonnées et sacrifient la conformité à un standard des PDF, comme les PDF archives, les PDF pour l'impression etc.

# **D'où vient le format PDF ?**

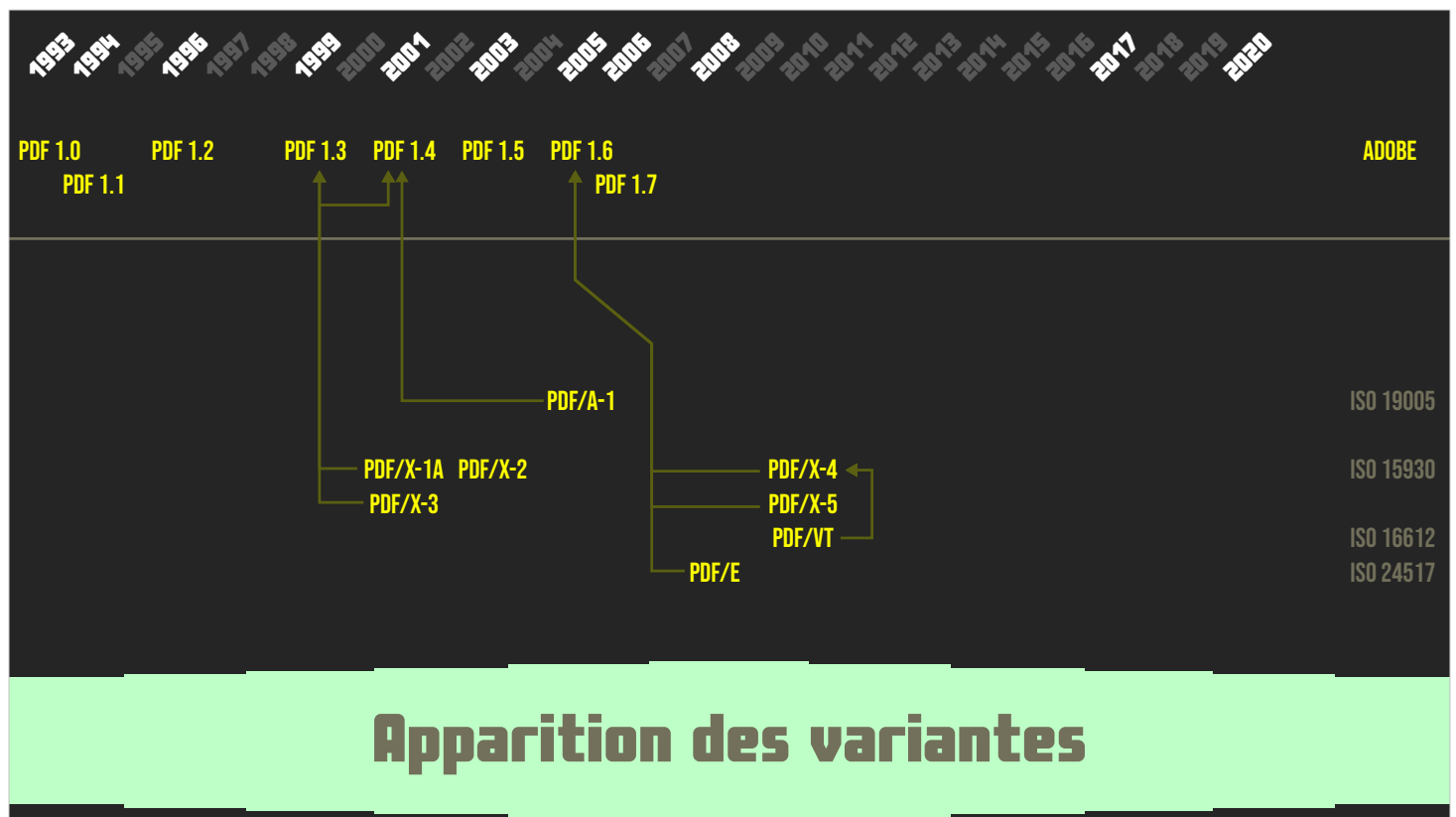
Nous allons faire un bref retour quelques années en arrière afin de mieux comprendre les raisons de ses structures.



En 1993, le project Camelot donne naissance au format PDF. C'est un format qui a accompagné l'apparition du web et s'est imprégné de l'air du temps pour lui ajouter régulièrement le support de nouvelles fonctionnalités et de formats d'images.

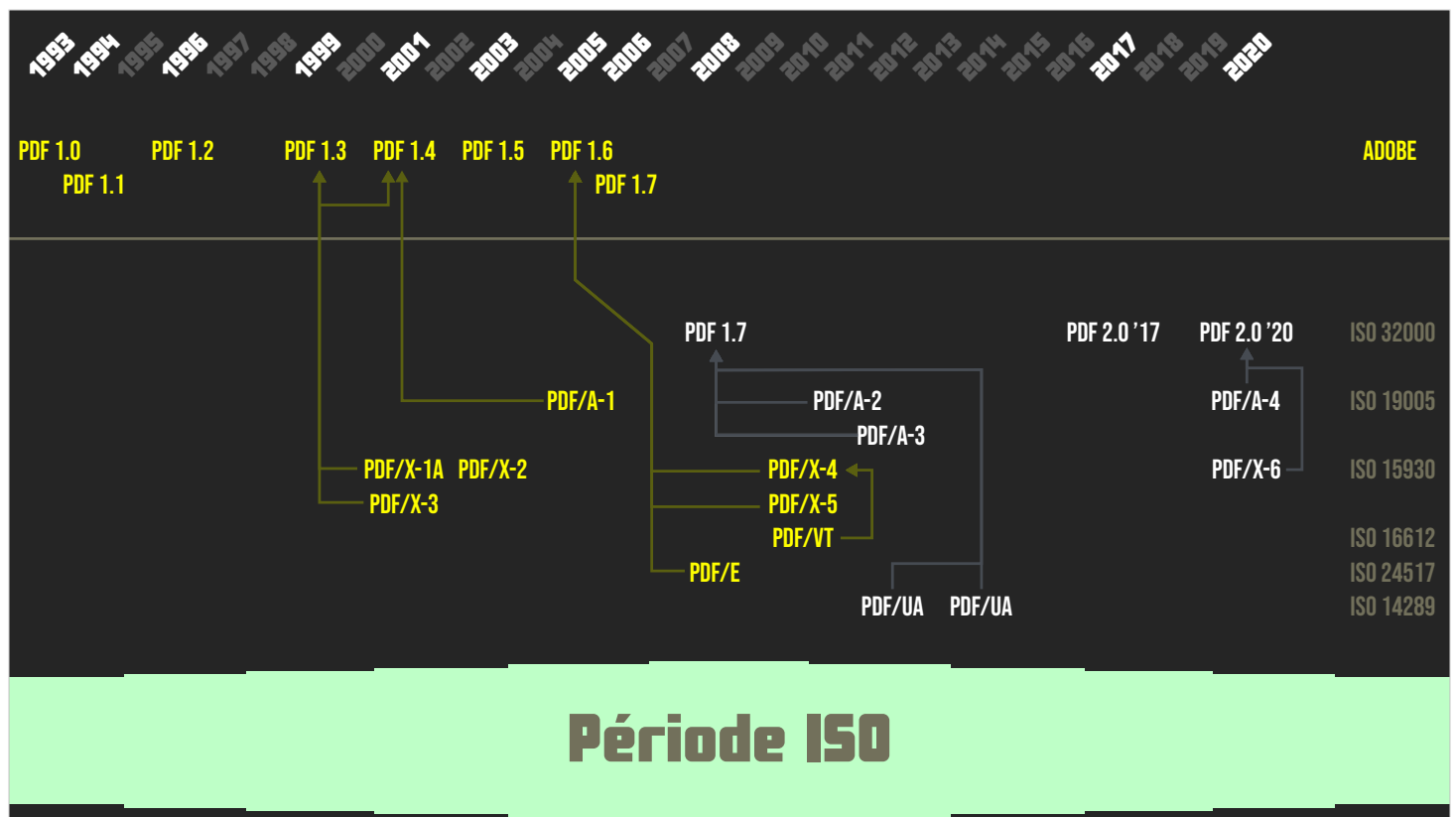
Adobe va le maintenir et le faire évoluer jusqu'en 2006.





## Apparition des variantes

À partir de 1999 on voit apparaître les variantes du format PDF, les fameuses PDF slash quelque chose. Elles vont permettre de résoudre des problématiques d'accessibilités (UA), d'impressions (X), d'archivage (A), de documentation technique (E) etc.



En 2008, la décision est prise de confier cette tâche à l'ISO afin d'en faire un standard : l'ISO 32000-1.

En 2017 sort la première version de PDF 2.0 ou ISO 32000-2, version qui sera révisée en 2020.

Hormis l'ajout de l'UTF-8, le but de la version 2.0 est de déprécier certaines fonctionnalités et de supprimer toute technologie propriétaire du format afin d'en faire un véritable standard.

# Comment est construit un PDF ?

On a donc un format qui a dû s'adapter aux évolutions techniques et aux nouveaux besoins tout en restant compatible. On va maintenant s'intéresser à la façon dont sont construits les PDF, condition sine qua non si on veut pouvoir les optimiser.



## UN FORMAT DE DONNÉES GÉNÉRIQUE

Le format PDF utilise un format de données générique qui pourrait être utilisé à d'autres fins. Dans l'esprit, on est très proche du format JSON mais plusieurs années avant.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Un langage inspiré de PostScript

C'est un langage inspiré de PostScript, le langage qui a fait les beaux jours d'Adobe dans les années 80.

PostScript est lui-même inspiré de Forth et d'autres langages fonctionnant au moyen d'une pile.

PDF en a encore les stigmates : pour chaque commande, la notation postfixée est utilisée ; l'opérateur est placé en dernier et non en premier. Mais, petite subtilité : il ne s'agit plus d'un langage à pile.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Commentaires

Le langage permet d'écrire des commentaires en commençant la ligne par un signe pourcentage.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Mots-clés

Il a quelques mots-clés permettant par exemple de créer des objets, des références, des tableaux, des chaînes de caractères etc.

Dans cet exemple, le code permet de créer un objet dictionnaire identifié par le numéro 4 et de génération 0. Le numéro de génération existe depuis le début mais il s'agit d'une fonctionnalité quasiment jamais utilisée.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Nombres

Il supporte des nombres entiers et flottants sans distinction forte entre les deux.



```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Chaînes de caractères

Il a deux types de chaînes de caractères.

Il y a les chaînes ASCII signalées par des parenthèses.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Chaînes hexadécimales

Les chaînes de caractères peuvent également être écrites au format hexadécimales.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Noms

Tous les caractères précédés d'un slash sont des noms, dont certains ont une utilisation bien définie en fonction du contexte où ils apparaissent.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Objets

Le mot-clé 'obj' permet d'attribuer une référence à un objet. Il est associé à un mot-clé 'endobj'.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Flux

Le mot-clé ‘stream’ permet d’embarquer des données brutes comme des images, des polices de caractères, du code graphique ou même des pièces jointes.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Dictionnaires

PDF dispose de dictionnaires. Ils sont repérés par des doubles signes inférieur/supérieur. C'est l'équivalent des accolades en JSON. Le nombre d'éléments est toujours pair puisqu'on est dans un système clé-valeur.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Clés de dictionnaires

Les dictionnaires sont indexés par des noms commençant par un slash. Et, tout comme en JSON, les structures peuvent être imbriquées.

Un certain nombre de noms sont obligatoires en fonction du contexte, d'autres optionnels. Il est également possible d'insérer ses propres noms tant qu'ils ne créent aucun conflit avec les noms standards.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Valeurs de dictionnaires

Les noms permettent de classer des valeurs dans le dictionnaires. Une valeur peut également être un nom.



```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Tableaux


Les tableaux/listes en PDF peuvent contenir n'importe quel type de données. Les types des données ainsi que leur nombre et leur ordre doivent cependant respecter le contexte dans lequel ils sont utilisés.

```
% Un commentaire
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Title     (Hello)
    /Creator   <48656C6C6F>
    /Resources << /Font 6 0 R >>
  >>
  stream ... endstream
endobj
```

## Références

Les références en PDF permettent de pointer sur d'autres objets, extérieurs à l'objet en cours.

# Table des références croisées



• Accès direct aux objets	xref		
• Offsets dans le fichier	1 7		
• Deux formats de table			
– historique, en clair	0000000028	00000	n
– Compressée, PDF ≥ 1.5	0000000257	00000	n
	0000000339	00000	n
	0000000433	00000	n
	0000000605	00000	n
	0000000926	00000	n
	0000000990	00000	n

Une table de références croisées historique apparaît telle qu'elle dans le fichier PDF

Certains objets sont à part et ont un format propre.

C'est le cas de la table des références croisées historique.

Elle permet un accès direct aux objets ce qui permet d'ouvrir ou d'imprimer des fichiers dont la taille dépasse la mémoire vive disponible. Si aujourd'hui la plupart des PDF peuvent être entièrement traités en mémoire, ce n'était pas le cas dans les années 90.

Elle a un format fixe (20 octets par ligne) afin de faciliter la recherche des offsets. Les offsets sont indiqués en décimal. Comme le format est fixe, cela limite à 10 Go les fichiers PDF utilisant ce format historique.

Elle est en clair mais il en existe une version compressée qui permet de s'affranchir de la limite des 10 Go.



## Trailer (annonce)

- Un dictionnaire non référencé
- Juste après la table des références croisées
- Références
  - Vers le catalogue du fichier PDF
  - Vers les métadonnées (titre, auteurs, dates de création...)
- Identifiant du PDF

L'annonce (ou trailer) accompagne la table des références croisées en clair. C'est le premier objet à consulter pour lire un PDF.

Il s'agit d'un dictionnaire qui n'est référencé nulle part ailleurs dans le PDF.

Elle accueille des références vers le catalogue du document ou vers les métadonnées.

Elle contient également un identifiant associé au fichier PDF.



# UN LANGAGE GRAPHIQUE

Pour tout ce qui est dessiné (dessin, texte...), PDF dispose d'un langage graphique basé sur le même formalisme.

```
% Début d'un bloc de texte
BT

% Dessine en noir en mode nuances de gris
0 g

% Utilise la fonte R7 à 96 points
/R7 96 Tf

% Définit la matrice de transformation du texte
1 0 0 1 170 270 Tm

% Écrit « Hello world! »
(Hello world!) Tj

% Fin du bloc de texte
ET
```

**Écrire « Hello world! »**

Voici un bout de code qui permet d'écrire « Hello world! » au milieu d'une page A4.

```
% Débute un bloc de texte
BT

% Dessine en noir en mode nuances de gris
0 g

% Utilise la fonte R7 à 96 points
/R7 96 Tf

% Définit la matrice de transformation du texte
1 0 0 1 170 270 Tm

% Écrit « Hello world! »
(Hello world!) Tj

% Fin du bloc de texte
ET
```

## Commandes graphiques

On retrouve des commandes.

Étant donnée leur utilisation intensive, celles-ci ont des noms abrégés :

- g pour « gray »
- Tf pour « Text font »
- Tm pour « Text matrix »
- Tj pour... « Show text » (oui, le nombre de lettres est limités)

```
% Débute un bloc de texte
BT

% Dessine en noir en mode nuances de gris
0 g

% Utilise la fonte R7 à 96 points
/R7 96 Tf

% Définit la matrice de transformation du texte
1 0 0 1 170 270 Tm

% Écrit « Hello world! »
(Hello world!) Tj

% Fin du bloc de texte
ET
```

## Paramètres

Le nombre et le type des paramètres dépend entièrement de la commande.

On peut y retrouver tous les types déjà vus précédemment comme des nombres (la grande majorité), des noms, des chaînes de caractères, des tableaux, des dictionnaires...



Hello world!

**Résultat du code « Hello world! »**

Et voici le résultat! (en admettant que la fonte R7 corresponde à l'Helvetica)



## DES FLUX ET DES FILTRES

Dans un PDF, il y a aussi des flux et des filtres.

# La ressource est dans le flux (stream)

- Images
  - Vectorielles
  - Matricielles
- Fichiers embarqués
  - Fichiers bureautiques
  - Métadonnées XMP
  - Vidéos
- Polices de caractères
  - TrueType
  - PostScript



Les flux vont recevoir tout ce qui est en dehors des possibilités du langage de base :

- Le langage graphique qu'on vient de voir.
- Les images matricielles (JPEG, PNG...)
- Les polices de caractères.
- Les fichiers embarqués.
- Les données multimédia.
- Les métadonnées au format XML.
- Etc.

# Chaque flux est « filtré »

- **Encodage**  
Ascii85, Hex
- **Compression sans perte**  
RLE, LZW, Flate, CCITTFax, JPEG2000
- **Compression avec perte**  
JPEG, JPEG2000, JBIG2
- **Prédicteurs pour certains filtres**  
TIFF, PNG
- **Les filtres peuvent se cumuler !**



Chaque flux est « filtré ».

Les filtres permettent d'encoder (Ascii85 ou hexadécimal), de compresser sans perte (RLE, LZW, Flate, CCITTFax, JPEG2000) ou avec perte (JPEG, JPEG2000, JBIG2) n'importe quel flux binaire.

Certains filtres peuvent recourir à des prédicteurs TIFF ou PNG.

Et on peut cumuler ces filtres !

## Exemple de flux filtré

```
36 0 obj
  <<
    /Length 40
    /Filter /ASCIIHexDecode
  >>
  stream
48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 21 >
endstream
endobj
```

Voici un exemple de flux filtré.

# Longueur du flux avant décodage

36 0 obj

<<

/Length 40

Le flux fait 40 octets

/Filter /ASCIIHexDecode

>>

stream

48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 21 >

endstream

endobj

Le flux brut, avant décodage, fait 40 octets.

42/111

## Encodage du flux

```
36 0 obj
  <<
    /Length 40
    /Filter /ASCIIHexDecode
  >>
  stream
  48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 21 >
endstream
endobj
```

Le flux a été encodé en hexadécimal

Il utilise le filtre « ASCIIHexDecode » qui, comme son nom l'indique, va transformer des nombres hexadécimaux en leur équivalent binaire.

Ce filtre a la particularité de permettre de n'avoir que des caractères ASCII dans un fichier PDF afin de faciliter la transmission de ce dernier dans n'importe quel mail.

Note : à l'époque de la création du format PDF, les échanges entre les différentes plateformes (PC, Mac, stations de travail Unix, mainframe, mini...) et systèmes d'exploitation (DOS, Windows, MacOS, OS/400, Unix...) étaient problématiques.

## Contenu du flux décodé

```
36 0 obj
  <<
    /Length 40
    /Filter /ASCIIHexDecode
  >>
  stream
48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 21 >
endstream
endobj
```

Le flux contient la  
chaîne de caractères  
«Hello, World!»

Une fois décodé, le flux contient la chaîne de caractères «Hello, World!»



# Lecture d'un PDF

Intéressons-nous maintenant à la lecture d'un fichier PDF. Les étapes présentées dans cette section sont exécutées par tout logiciel manipulant des fichiers PDF, que ce soient des visionneuses ou des logiciels d'édition, de manipulation.

# Un format conçu en 1993 pour...

- La transmission
  - Fun fact : un PDF peut être **entièrement** encodé en ASCII !
- Des tailles de fichier énormes
  - PDF < 1.5 : maximum de ~9,3 Gio
  - PDF ≥ 1.5 : pas de limite théorique
- La mise à jour incrémentale
  - CRUD = Simple ajout à la fin du fichier



Il faut garder à l'esprit que le PDF est un format conçu avec plusieurs contraintes en tête.

Le PDF doit pouvoir être entièrement codé en ASCII.

Il doit permettre de lire des fichiers énormes sans nécessiter une grosse configuration mémoire.

Il a aussi été conçu pour la mise à jour incrémentale : pour mettre à jour un PDF, il suffit juste d'ajouter les bonnes structures en fin de fichier.

À titre indicatif, en 1990 :

- les PC avaient entre 1 et 4 Mo de RAM, 8 Mo pour les grosses configurations
- Les cartes graphiques entre 256 Ko et 1 Mo



ENTÊTE

## Vérification de l'entête

La lecture commence à l'entête afin de vérifier qu'il s'agit bien d'un fichier PDF.

# L'entête d'un PDF

- **Signature %PDF- {version}**

1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7 et 2.0

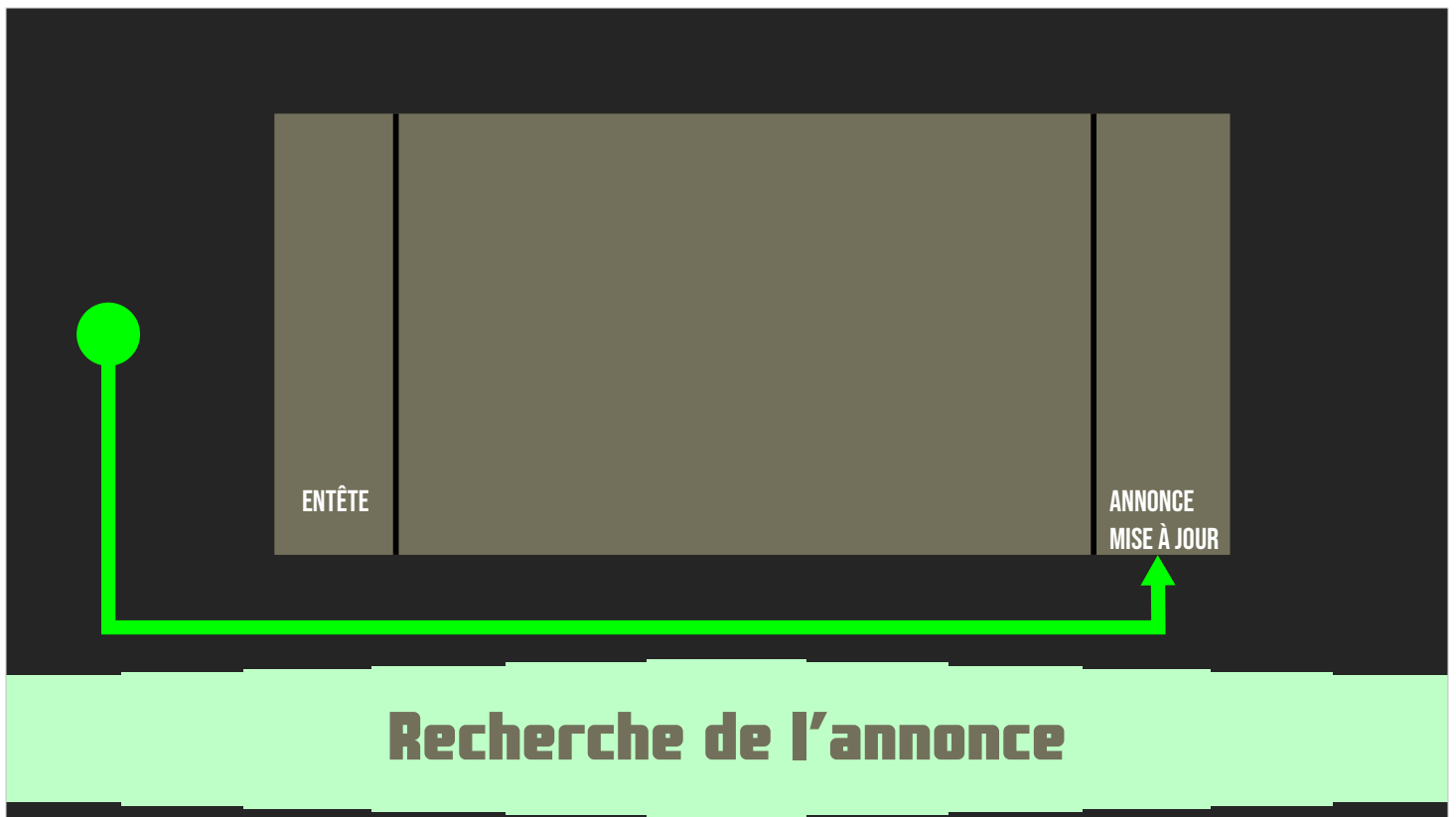
- **Commentaire «détrompeur» (optionnel)**

4 caractères 8 bits non ASCII (> 127)

00000000	25 50 44 46 2d 31 2e 37 0a 25 e2 e3 cf d3 0d 0a	%PDF-1.7.%.....
00000010	39 30 37 39 33 20 30 20 6f 62 6a 0a 3c 3c 2f 4c	90793 0 obj.<</L
00000020	69 6e 65 61 72 69 7a 65 64 20 31 2f 4c 20 31 34	linearized 1/L 14
00000030	37 32 31 30 38 38 2f 4f 20 39 30 37 39 35 2f 45	721088/0 90795/E
00000040	20 31 30 30 36 33 39 2f 4e 20 31 30 30 33 2f 54	100639/N 1003/T
00000050	20 31 34 37 30 39 36 34 36 2f 48 20 5b 20 33 34	14709646/H [ 34
00000060	31 34 20 39 39 35 35 5d 3e 3e 0a 65 6e 64 6f 62	14 9955]>>.endob

Pour lire un PDF, une visionneuse va tout d'abord rechercher la signature qui commence par %PDF- suivi du numéro de version du format utilisé.

Il est aujourd'hui suivi d'un détrompeur permettant aux logiciels qui tentent de déterminer s'il s'agit d'un fichier ASCII ou binaire de ne pas se fourvoyer (beaucoup de fichiers PDF n'ont pas de caractères non-ASCII à leur début).



Ensuite, il faut se déplacer à la toute fin du fichier pour découvrir l'annonce.

# Repérer les références croisées

- **Fin de fichier %EOF**

Peut se trouver n'importe où dans les 1024 derniers octets !

- **Offset de la table des références croisées startxref**

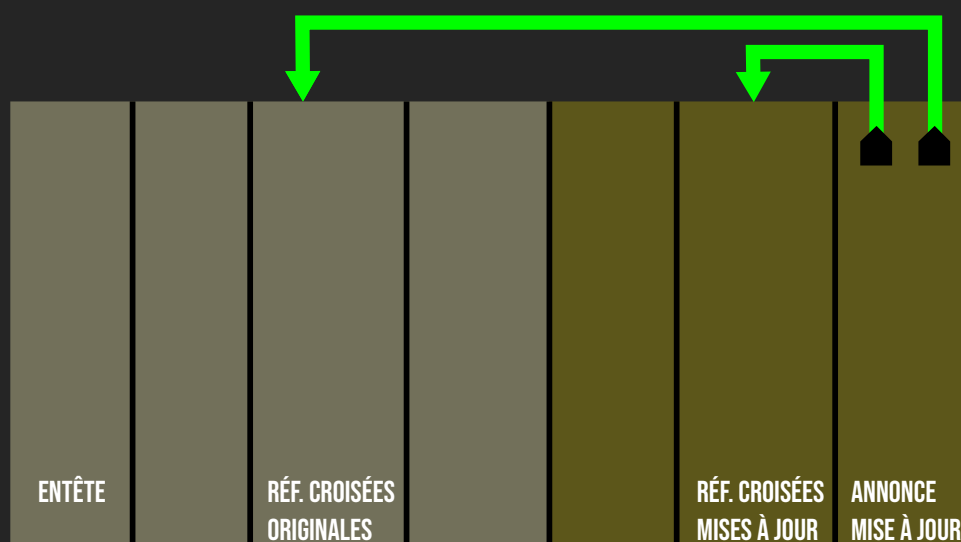
En base 10 !

00e09fe0	f1 0a 42 bc 82 78 05 f1	0a 42 bc 82 78 05 f1 0a	..B..x...B..x...
00e09ff0	42 bc 82 78 05 f1 0a 42	bc 82 78 05 f1 0a c2 e9	B..x...B..x.....
00e0a000	7a 55 fc c1 b7 ab 4f d2	7a 90 28 07 f1 ea af ba	zU....0.z.(.....
00e0a010	dd be f3 ff 06 00 79 9f	aa 6a 0d 65 6e 64 73 74	.....y...j.endst
00e0a020	72 65 61 6d 0d 65 6e 64	6f 62 6a 0d 73 74 61 72	ream.endobj.star
00e0a030	74 78 72 65 66 0a 31 31	36 0a 25 25 45 4f 46 0a	txref.116.%%EOF.

Il faut ensuite aller à la toute fin du fichier et y rechercher le marqueur de fin de fichier %EOF qui peut se trouver n'importe où dans les 1024 derniers octets.

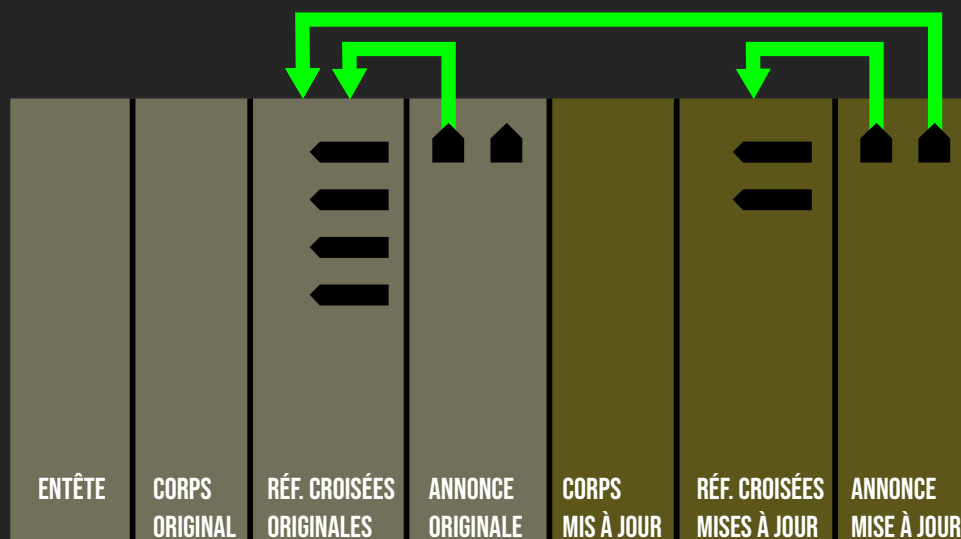
Juste avant ce marqueur se trouve un bloc startxref qui permet de savoir où, dans le fichier, se trouve la table des références croisées.

Dans cet exemple, elle se trouve au 116<sup>e</sup> octet. Donc quasiment au début du fichier.



## Recherche des références croisées

L'annonce indique la position de la table des références croisées et de son éventuelle version antérieure.



## Décodage des références croisées

Il faut ensuite décoder les tables de références croisées.



# Décoder les références croisées

00000070	6a 0d 20 0d	39 30 38 32	34 20 30 20	6f 62 6a 0d	j. .90824 0 obj.
00000080	3c 3c 2f 44	65 63 6f 64	65 50 61 72	6d 73 3c 3c	<</DecodeParms<<
00000090	2f 43 6f 6c	75 6d 6e 73	20 35 2f 50	72 65 64 69	/Columns 5/Predi
000000a0	63 74 6f 72	20 31 32 3e	3e 2f 46 69	6c 74 65 72	ctor 12>>/Filter
000000b0	2f 46 6c 61	74 65 44 65	63 6f 64 65	2f 49 44 5b	/FlateDecode/ID[
000000c0	3c 32 42 35	35 31 44 32	41 46 45 35	32 36 35 34	<2B551D2AFE52654
000000d0	34 39 34 46	39 37 32 30	32 38 33 43	46 46 31 43	494F9720283CFF1C
000000e0	34 3e 3c 33	43 44 41 38	42 42 36 44	35 38 33 34	4><3CDA8BB6D5834
000000f0	45 34 31 41	35 45 32 41	41 31 36 43	33 35 45 34	E41A5E2AA16C35E4
00000100	43 34 37 3e	5d 2f 49 6e	64 65 78 5b	39 30 37 39	C47>]/Index[9079
00000110	33 20 31 30	31 34 5d 2f	49 6e 66 6f	20 39 30 37	3 1014]/Info 907
00000120	39 32 20 30	20 52 2f 4c	65 6e 67 74	68 20 31 38	92 0 R/Length 18
00000130	35 2f 50 72	65 76 20 31	34 37 30 39	36 34 37 2f	5/Prev 14709647/
00000140	52 6f 6f 74	20 39 30 37	39 34 20 30	20 52 2f 53	Root 90794 0 R/S
00000150	69 7a 65 20	39 31 38 30	37 2f 54 79	70 65 2f 58	ize 91807/Type/X
00000160	52 65 66 2f	57 5b 31 20	33 20 31 5d	3e 3e 73 74	Ref/W[1 3 1]>>st

*Et si on simplifiait la lecture de ce dump hexadécimal ?*

Au 116<sup>e</sup> octet (0x74 en hexadécimal) débute un objet numéro 90824 0.

Voyons donc ce qu'il contient dans un format plus lisible.

# Un flux de références croisées

```
90824 0 obj <<
```

```
  /Type      /Xref
```

```
  /Size      91807
```

```
  /Index     [90793 1014]
```

```
  /Filter    /FlateDecode
```

```
  /DecodeParms <</Columns 5 /Predictor 12>>
```

```
  /Length    185
```

```
  /W         [1 3 1]
```

```
  /Info      90792 0 R
```

```
  /Root      90794 0 R
```

```
  /Prev      14709647
```

```
  /ID        [...]
```

```
>>
```

Cette ligne indique des références croisées compressées

Il s'agit d'un objet numéroté 90824 qui contient un dictionnaire (d'après le double chevron).

La clé /Type nous indique qu'il s'agit d'un objet de type flux de références croisées (XRefStream dans la terminologie PDF).

# Numéro de départ des futurs objets

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index      [90793 1014]
  /Filter      /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length      185
  /W          [1 3 1]
  /Info        90792 0 R
  /Root        90794 0 R
  /Prev        14709647
  /ID          [...]
>>
```

Utilisé pour de potentielles futures mises à jour du PDF

Le champ /Size, contrairement à ce que son nom laisse suggérer, ne donne pas la taille mais le numéro à utiliser comme point de départ de numérotation des nouveaux objets lors d'une éventuelle mise à jour incrémentale.

# Numéro du 1<sup>er</sup> objet et nombre d'objets

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index      [90793 1014]
  /Filter     /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length    185
  /W         [1 3 1]
  /Info       90792 0 R
  /Root       90794 0 R
  /Prev       14709647
  /ID         [...]
>>
```

90793 + 1014 = 91807

Le champ /Index donne deux valeurs :

- La première (90793) indique le numéro d'objet associé à la toute première référence croisée de la table.
- La deuxième (1014) indique le nombre d'objets contenus dans la table.

Et, comme le monde est bien fait,  $90793 + 1014 = 91807$ , soit le nombre donné par la clé /Size.

# Flux compressé par Flate (Zlib)

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index     [90793 1014]
  /Filter     /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length    185
  /W         [1 3 1]
  /Info      90792 0 R
  /Root      90794 0 R
  /Prev      14709647
  /ID        [...]
>>
```

La clé /Filter nous indique que le flux est compressé selon l'algorithme Flate (le même utilisé dans Gzip).

Il faudra donc décompresser le flux avant de pouvoir l'exploiter.

# Prédicteur Up et entrée = 5 octets

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index     [90793 1014]
  /Filter    /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length    185
  /W         [1 3 1]
  /Info      90792 0 R
  /Root      90794 0 R
  /Prev      14709647
  /ID        [...]
>>
```

1 entrée de la table = 5 octets  
Prédicteur utilisé = «Up»

La clé /DecodeParms nous indique que le prédicteur 12 a été utilisé avant la compression afin d'optimiser celle-ci.

12 correspond au prédicteur «PNG Up» qui est présenté plus loin dans cette présentation.

La clé /Columns indique que le prédicteur a travaillé sur des données groupées par bloc de 5 octets.

# Flux compressé = 185 octets

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index     [90793 1014]
  /Filter     /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length    185
  /W         [1 3 1]
  /Info       90792 0 R
  /Root       90794 0 R
  /Prev       14709647
  /ID         [...]
>>
```

La clé /Length indique que le flux compressé fait 185 octets.

# Taille des champs en octets

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index      [90793 1014]
  /Filter     /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length    185
  /W         [1 3 1]
  /Info       90792 0 R
  /Root       90794 0 R
  /Prev       14709647
  /ID         [...]
>>
```

1 octet pour le type de l'entrée  
3 octets pour un offset/n° d'objet  
1 octet pour la génération/index

La table de références croisées contient des enregistrements ayant chacun 3 champs.

La clé /W (pour « Widths ») indique les largeurs en octets de chacun de ces champs :

- Le premier champ fait un octet de large.
- Le deuxième trois.
- Le troisième un.



# Métadonnées du document

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index     [90793 1014]
  /Filter     /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length    185
  /W         [1 3 1]
  /Info      90792 0 R
  /Root      90794 0 R
  /Prev      14709647
  /ID        [...]
>>
```

Les métadonnées sont  
dans l'objet 90792

La clé /Info contient la référence vers l'objet contenant les métadonnées du fichier PDF.

# Catalogue du document

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index      [90793 1014]
  /Filter     /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length     185
  /W          [1 3 1]
  /Info       90792 0 R
  /Root       90794 0 R
  /Prev       14709647
  /ID         [...]
>>
```

Le catalogue est dans  
l'objet 90794

La clé /Root contient la référence vers l'objet contenant le catalogue du fichier PDF.

# Offset des références précédentes

```
90824 0 obj <<
  /Type      /Xref
  /Size      91807
  /Index      [90793 1014]
  /Filter     /FlateDecode
  /DecodeParms <</Columns 5 /Predictor 12>>
  /Length     185
  /W          [1 3 1]
  /Info       90792 0 R
  /Root       90794 0 R
  /Prev       14709647
  /ID         [...]
>>
```

Une précédente table de références croisées existe à l'offset 14709647. La table courante est prioritaire sur elle.

La clé /Prev (pour « Previous »), si elle est présente, donne l'offset de la précédente table de références croisées.

Chaque table de références croisées contient un sous-ensemble de toutes les références contenues dans le fichier PDF.

Dans le cas de doublons, c'est la table la plus récente qui l'emporte.

## Trouver l'offset du catalogue

- Décoder la table des références croisées

- 3 colonnes par entrée

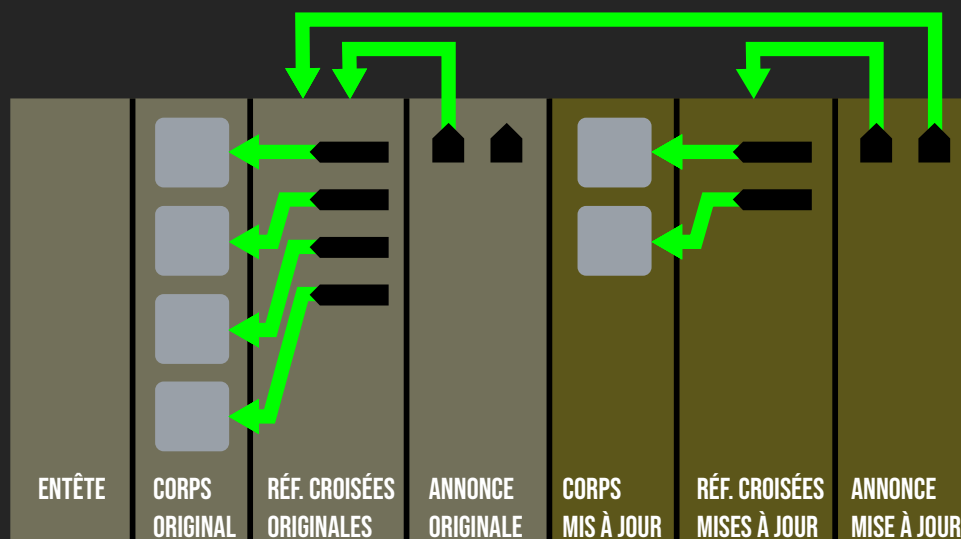
- Type d'entrée: 1
- Offset: 0x3439 ou 13369
- Numéro de génération: 0

90793	01	000010	00
90794	01	003439	00
90795	01	00363E	00
90796	01	003E53	00
90797	01	004228	00
...			
90825	02	0162AC	00
90826	02	0162AC	01
90827	02	0162AC	02
90828	02	0162AC	03
90829	02	0162AC	04
...			

Une fois la table des références croisées décodée, on retrouve nos 3 colonnes de 5 octets chacune (le numéro d'objet n'est pas stocké dans la table, il doit être calculé).

Le premier champ donne le type d'entrée. 1 signifie que le deuxième champ contient l'offset de l'objet dans le fichier. 2 signifie que le deuxième champ contient le numéro de l'objet dans lequel l'objet qui nous intéresse est compressé.

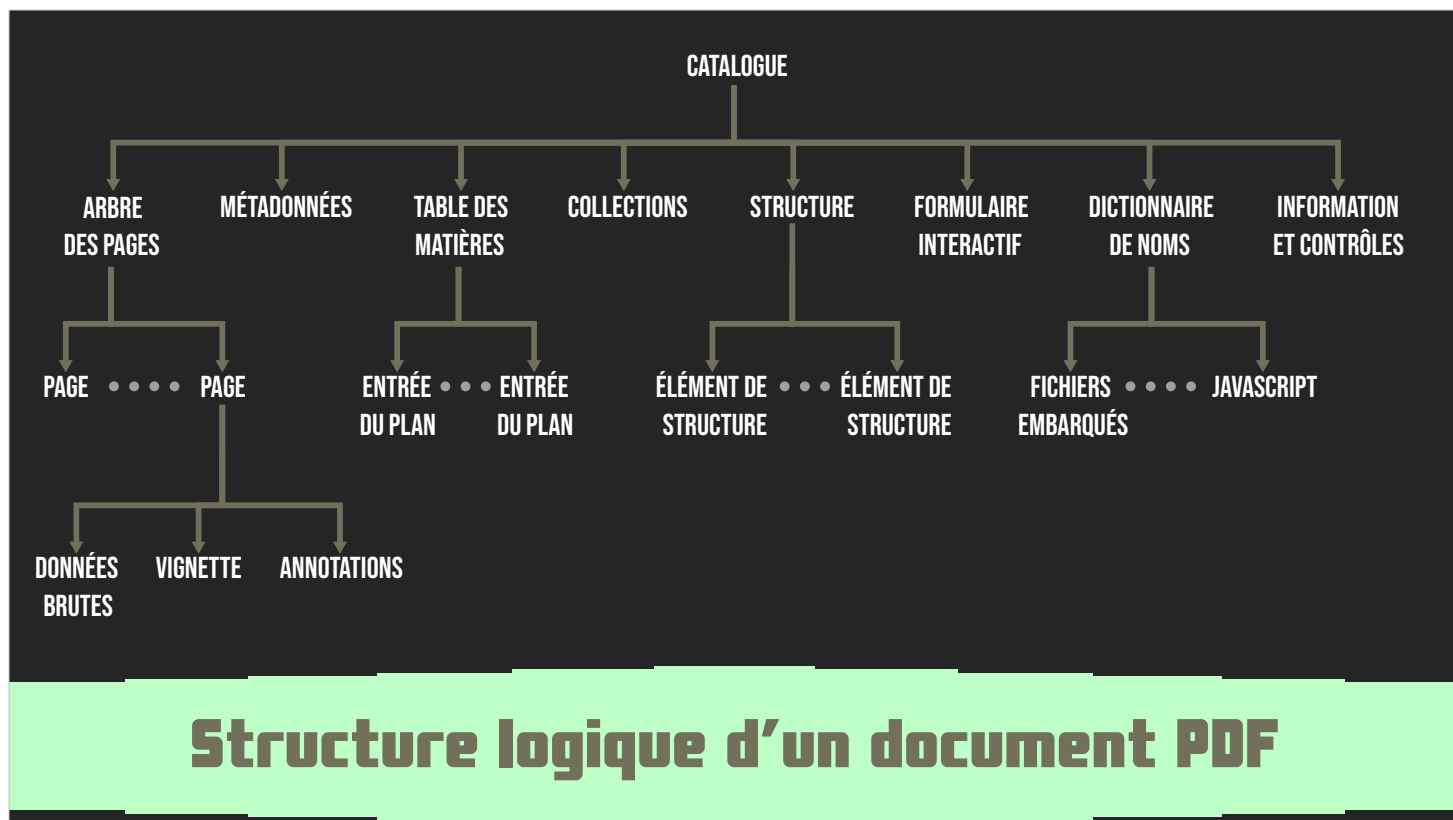
Si on veut trouver le catalogue dont le numéro est le 90794, il faut donc se rendre à l'offset 13369 (ou 0x3439).



## Recherche des objets

Les références croisées donne les offsets de chacun des objets.

Ce système permet la mise à jour par simple ajout de données en fin de fichier.



Et ainsi de suite jusqu'à parcourir toute la structure du PDF pour en extraire les informations nécessaires pour faire le rendu d'une ou plusieurs pages.

# Optimisation !

Regardons maintenant comment on peut optimiser tout ça !



# LES CHAÎNES DE CARACTÈRE



Intéressons-nous tout d'abord aux chaînes de caractères.



## 2 façons de coder des chaînes

- Chaînes littérales

(A string)

((word))

(P)

Pas besoin d'échappement  
si les parenthèses sont  
équilibrées

- Chaînes hexadécimales

<4120737472696E67>

<28 77 6F 72 64 29>

<5>

Le zéro terminal est  
optionnel, 5 = 50

Il y a 2 façons de coder des chaînes de caractères : sous forme de chaînes littérales encadrées par des parenthèses ou sous forme de chaînes hexadécimales encadrées par des inférieurs/supérieurs.

## 2 encodages, 5 types de chaînes

Type	Caractères	Littérale	Hexadécimale
Chaîne d'octets <i>byte string</i>	256	n + échappements	2n
ASCII <i>pas d'accents!</i>	127	n + échappements	2n
UTF-16BE <i>big endian</i>	Unicode	$2+2n \leq \text{taille} \leq 2+4n$ + échappements	$4+4n \leq \text{taille} \leq 4+8n$
UTF-8 <i>PDF 2.0+</i>	Unicode	$3+n \leq \text{taille} \leq 3+4n$ + échappements	$6+2n \leq \text{taille} \leq 6+8n$
PDFDocEncoded <i>~Latin-1</i>	256	n + échappements	2n

De nombreux générateurs de PDF produisent des chaînes hexadécimales. Celles-ci occupent deux fois plus de place que les chaînes littérales. Une optimisation consiste à opérer une conversion.



## SUPPRIMER LE SUPERFLU



Il y a plusieurs grands principes quand on veut réduire le poids d'un fichier.

Tout d'abord, il faut supprimer le superflu : tout ce qui n'est pas nécessaire pour faire le rendu d'un document à l'identique.

# Que supprimer ?



- **Éléments inutiles**
  - Retours chariots
  - Espaces
  - Commentaires
  - Caractères inutiles
- **Précision inutile**
- **Objets inutilisés**
- **Éléments**
  - Cachés
  - Hors-cadres
  - Dupliqués
- **Instructions redondantes**
- **Valeurs identiques aux valeurs par défaut**

On va pouvoir supprimer tous les éléments du langage utiles généralement pour une lecture humaine mais inutiles pour l'ordinateur.

On va voir aussi que certains logiciels poussent la précision de certaines valeurs au-delà de l'utile.

Certains objets sont inutilisés, par exemple à cause d'une mise à jour incrémentale.

Des éléments graphiques non visibles ou redondants peuvent avoir été générés par les logiciels.

PDF a également la notion de valeurs par défaut, on peut gagner de l'espace de ce côté-là aussi.



## Code non minifié

```
4 0 obj
  <<
    /Type      /Page
    /Parent    3 0 R
    /MediaBox  [0 0 842 595]
    /Contents  5 0 R
    /Resources << /Font 6 0 R >>
  >>
endobj
```

Voici un morceau de code parfaitement valide mais présenté de façon à ce que nous, humains, puissions le lire facilement.

# 148 octets dont 45 inutiles

00000000	34	20	30	20	6f	62	6a	0a	20	20	3c	3c	0a	20	20	20	4 0 obj. <<.
00000010	20	2f	54	79	70	65	20	20	20	20	20	20	2f	50	61	67	/Type /Pag
00000020	65	0a	20	20	20	20	2f	50	61	72	65	6e	74	20	20	20	e. /Parent
00000030	20	33	20	30	20	52	0a	20	20	20	20	2f	4d	65	64	69	3 0 R. /Medi
00000040	61	42	6f	78	20	20	5b	30	20	30	20	38	34	32	20	35	aBox [0 0 842 5
00000050	39	35	5d	0a	20	20	20	20	2f	43	6f	6e	74	65	6e	74	95]. /Content
00000060	73	20	20	35	20	30	20	52	0a	20	20	20	20	2f	52	65	s 5 0 R. /Re
00000070	73	6f	75	72	63	65	73	20	3c	3c	20	2f	46	6f	6e	74	sources << /Font
00000080	20	36	20	30	20	52	20	3e	3e	0a	20	20	3e	3e	0a	65	6 0 R >>. >>.e
00000090	6e	64	6f	62	6a												ndobj

*Un dump hexadécimal permet de mieux se rendre compte*

En le regardant à la loupe, on s'aperçoit que 45 octets sur les 148 du bloc sont parfaitement inutiles à une lecture par un ordinateur.

Les logiciels générant des PDF ont tendance à insérer des espaces entre chaque élément, même si cela n'est pas nécessaire, pour des raisons de simplicité la plupart du temps.

Il n'y a pas besoin d'espace entre les symboles de ponctuation (comme les signes plus petit que et plus grand que, les accolades, les parenthèses, les crochets, les slash) et les autres caractères.

## Précision inutile, exemple 1

- Ex.: 0.14509   0.14509   0.14509   RG  

rouge   vert   bleu   couleur du tracé
- 5 chiffres de précision
  - 100 000 niveaux de rouge, de vert et de bleu
  - 1 000 000 000 000 000 de couleurs possibles!
  - L'œil humain voit 10 000 000 de couleurs et 30 niveaux de gris  
<https://royalsocietypublishing.org/doi/full/10.1098/rsif.2012.0601>

Le PDF travaille souvent avec des nombres flottants. Il n'y a d'ailleurs pas de réelles distinction entre un nombre entier et un nombre à virgule dans ce langage.

Les logiciels ne semblent pas tous évaluer le niveau de précision nécessaire à chaque valeur.

Dans cet exemple, la sélection de la couleur de tracé utilise 5 chiffres de précision.

Pour une couleur de tracé.

Cela correspond à 100 000 niveaux de rouge, de vert et de bleu, soit un million de milliards de couleurs possibles.

Sachant que l'œil humain ne peut discerner que 10 millions de couleurs et 30 niveaux de gris...

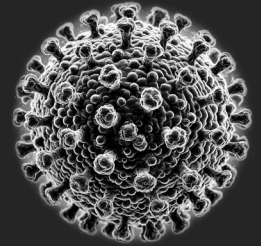
Les octets gagnés sont importants car le code utilisant ce type de fonction peut contenir énormément de commandes.

## Précision inutile, exemple 2

• Ex.: 0 0.028 793.672 446.428 re  
x y largeur hauteur trace un rectangle

- L'unité utilisée est le point

- 1 point, 1/72<sup>e</sup> pouce..... 0,353 mm
- 0,001 point..... 0,000 353 mm
- Virus de la grippe..... 0,000 100 mm



Autre exemple de précision inutile, cette fois sur les coordonnées utilisées pour tracer des éléments sur une page. Ici pour tracer un rectangle.

L'unité native utilisée par le PDF est le point qui correspond à 1/72<sup>e</sup> de pouce soit à peu près 353 micromètres ou 0,353 millimètres.

Un millième de point correspond à 353 nanomètres, soit du même ordre de grandeur que des virus.

Un niveau de précision hautement utile...



## Valeurs par défaut inutiles

```
4 0 obj
<<
  /Length 37
  /Filter [/ASCIIHexDecode /FlateDecode]
  /DecodeParms [
    null
    << /Predictor 11 /Columns 26 /Components 8 /Colors 1 >>
  ]
>>
stream
78 da 63 74 64 c4 05 00 08 15 00 5c >
endstream
endobj
```

Certains logiciels peuvent mettre des valeurs qui correspondent déjà aux valeurs par défaut.

Dans cet exemple, la valeur par défaut de la clé Components est 8 et celle de Colors 1.

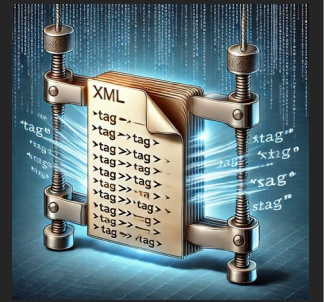
## Valeurs par défaut supprimées

```
4 0 obj
<<
  /Length 37
  /Filter [/ASCIIHexDecode /FlateDecode]
  /DecodeParms [
    null
    << /Predictor 11 /Columns 26 >>
  ]
>>
stream
78 da 63 74 64 c4 05 00 08 15 00 5c >
endstream
endobj
```

On peut donc les supprimer sans crainte.

# Optimiser les données XML

- Suppression des espaces inutiles
- Optimisation des JPEG embarqués  
Un JPEG, encodé en base64, dans du XML, dans un flux compressé, dans un PDF...
- Renommage des espaces de nom



Les évolutions du format PDF poussent à l'utilisation du format XML pour le stockage des métadonnées plutôt qu'à des structures traditionnelles du PDF.

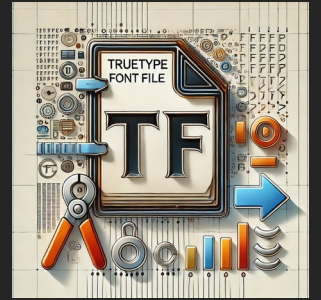
Un fichier XML peut être optimisé en supprimant les espaces inutiles, en optimisant les JPEG embarqués (vignettes) et en renommant les différents espaces de noms utilisés.

# Optimiser les fontes True Type

- Suppression des informations de «hinting»

Des informations dédiées à l'amélioration du rendu sur écran mais souvent ignorées

- Suppression des commentaires



Les polices True Type embarquent souvent des informations de hinting qui sont dédiées à l'amélioration du rendu sur écran. Celles-ci sont souvent ignorées en fonction des plateformes et des visionneuses utilisées.

Les polices True Type peuvent également embarquer des commentaires inaccessibles aux utilisateurs.

# Supprimer les éléments inutiles

- Tout élément ou image
  - Dupliqué
  - Utilisant des masques de découpe
  - Sortant du cadre
  - Masqué par d'autres éléments
  - Utilisant une résolution  $\geq 300$  ppp
- Un problème complexe !



Pour alléger un PDF, on peut également supprimer les éléments qui n'ont aucune incidence sur le rendu final du document.

Cela inclut les éléments qui ont été dupliqués, ceux utilisant des masques de découpe pour lesquels l'image originale reste enregistrée dans le PDF, les images sortant du cadre ou celles masquées par d'autres éléments.

On peut également éliminer un surplus d'information pour les images ayant une résolution dépassant les besoins du rendu.

Ce n'est toutefois pas un problème simple !

# PowerPoint vs Impress

- Comparatif des stratégies face à des images
- Test effectué sur

– PowerPoint	Microsoft Office 365	16.0.16904.40516
– Impress	LibreOffice	24.8.0.3



Pour illustrer la complexité de la problématique, prenons un exemple avec les PDF générés par PowerPoint et par Impress.

On va comparer les stratégies de chacun quant à la gestion des images.

La plupart des hommes ne s'étonnent point assez. En présence des plus grands phénomènes, des inventions les plus admirables, on les voit trop souvent indifférents, impassibles. C'est le propre de la matière d'être impassible, et non pas de l'esprit. Ceux dont la curiosité est toujours en éveil, qui aiment à s'expliquer ce qu'ils voient, qui recherchent les causes, ceux-là seuls parviennent à s'instruire, à s'éclairer, à augmenter leurs jouissances intellectuelles, et peuvent, s'ils sont doués de quelque supériorité, contribuer à l'avancement des sciences et de leurs applications, c'est-à-dire au progrès du bien-être de leurs semblables et de la civilisation. Voici, par exemple, les chemins de fer et le télégraphe électrique qui ne datent que de peu d'années : on s'y est déjà si bien habitué qu'il ne semble que ces merveilleuses inventions aient existé de tout...



## Diapo 1, JPEG rogné

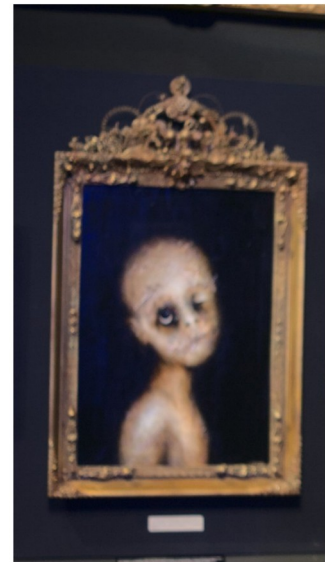
Voici la première diapositive avec le JPEG rogné...



## Diapo 2, JPEG complet

... la deuxième diapositive avec le JPEG complet

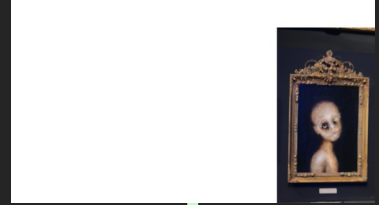




### **Diapo 3, JPEG débordant du cadre**

... et la troisième diapositive avec le JPEG débordant du cadre.

La plupart des hommes ne s'étonnent point assez. En présence des plus grands phénomènes, des inventions les plus admirables, on les voit trop souvent indifférents, impassibles. C'est le propre de la matière d'être impassible, et non pas de l'esprit. Ceux dont la curiosité est toujours en éveil, qui aiment à s'expliquer ce qu'ils voient, qui recherchent les causes, ceux-là seuls parviennent à s'instruire, à s'éclairer, à augmenter leurs jouissances intellectuelles, et peuvent, s'ils sont doués de quelque supériorité, contribuer à l'avancement des sciences et de leurs applications, c'est-à-dire au progrès du bien-être de leurs semblables et de la civilisation. Voici, par exemple, les chemins de fer et le télégraphe électrique qui ne datent que de peu d'années : on s'y est déjà si bien habitué qu'il ne semble que ces merveilleuses inventions aient existé de tout...

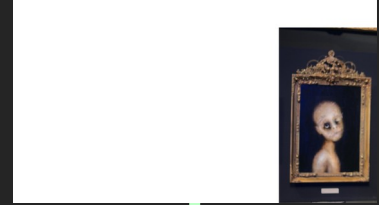


**PDF généré par PowerPoint**

Du côté de PowerPoint, quand vous rognez une image, le PDF ne contiendra que la version rognée. Par contre, les images débordant ne sont pas exportées rognées.

PowerPoint fait tout de même preuve d'intelligence en ne dupliquant pas l'image originale dans le PDF.

La plupart des hommes ne s'étonnent point assez. En présence des plus grands phénomènes, des inventions les plus admirables, on les voit trop souvent indifférents, impassibles. C'est le propre de la matière d'être impassible, et non pas de l'esprit. Ceux dont la curiosité est toujours en éveil, qui aiment à s'expliquer ce qu'ils voient, qui recherchent les causes, ceux-là seuls parviennent à s'instruire, à s'éclairer, à augmenter leurs jouissances intellectuelles, et peuvent, s'ils sont doués de quelque supériorité, contribuer à l'avancement des sciences et de leurs applications, c'est-à-dire au progrès du bien-être de leurs semblables et de la civilisation. Voici, par exemple, les chemins de fer et le télégraphe électrique qui ne datent que de peu d'années : on s'y est déjà si bien habitué qu'il ne semble que ces merveilleuses inventions aient existé de tout...



**PDF généré par Impress**

Pour Impress, la stratégie est différente : la version originale de l'image rognée est conservée dans le PDF.

À l'instar de PowerPoint, Impress ne duplique pas l'image originale dans le PDF généré.



# Les stratégies de génération du PDF

- Rognage
  - PowerPoint **rogné**
  - Impress **masque**
- Débordement
  - Aucune gestion du débordement
- Duplication
  - Pas de duplication mais rogner = dupliquer
- Résolution
  - Adaptation sauf pour Office 365 web

Il n'y a pas de stratégie gagnante à tous les coups. Dans notre exemple, c'est la stratégie d'Impress qui est la plus efficace. Mais si nous n'avions eu qu'une seule diapositive avec une image rognée, c'est la stratégie de PowerPoint qui l'aurait emporté.



# OPTIMISER LE CODE GRAPHIQUE



Chaque page d'un PDF utilise du code graphique.



## Tout un programme !

- Espaces inutiles
- Saut de ligne → espace
- Commandes ineffectives
- Commandes plus courtes
- Tri de commandes
- Fusion de commandes
- Sauvegardes et restauration inutiles
- Chaînes de caractères
- Noms des ressources
- Précision des nombres
- Factorisation
- Etc.

À l'instar de la minification/offuscation du JavaScript, il est possible d'utiliser de nombreuses astuces pour optimiser le code comme la suppression des espaces inutiles, la suppression de commandes sans effet, de fusionner des commandes, de réduire la précision des nombres etc.

Les diapositives suivantes présentent quelques exemples d'optimisation.

## Utilisation d'opérateurs dédiés

```
MoveTo( 0, 0)
```

```
LineTo( 0, 20)
```

```
LineTo(50, 20)
```

```
LineTo(50, 0)
```

```
LineTo( 0, 0)
```

```
CloseSubpath
```

```
FillPath
```

```
Rectangle(0, 0, 50, 20)
```

```
FillPath
```

*(Du pseudo-code est utilisé pour simplifier la compréhension)*

Le PDF dispose de quelques opérateurs permettant de simplifier l'écriture.

Par exemple, il existe un opérateur traçant un rectangle. De façon surprenante, de nombreux générateurs de PDF n'utilisent pas cet opérateur et tracent des rectangles en utilisant des droites.

A contrario, il n'existe pas d'opérateur traçant des cercles ou des ovales. Celles-ci doivent l'être avec des courbes de Bézier.

## Opérations inutiles

SaveGS

Rectangle(0, 0, 5, 2)

FillPath

SaveGS

MoveTo( 0, 0)

LineTo( 0, 20)

StrokePath

RestoreGS

RestoreGS

SaveGS

Rectangle(0, 0, 5, 2)

FillPath

MoveTo( 0, 0)

LineTo( 0, 20)

StrokePath

RestoreGS

*(Du pseudo-code est utilisé pour simplifier la compréhension)*

Le PDF dispose d'un système de pile pour sauvegarder l'état du moteur graphique. Cela autorise l'import d'éléments sans nécessiter leur réécriture pour prendre en compte une éventuelle transformation (le décalage est une transformation).

Les générateurs de PDF usent et abusent souvent des fonctions de sauvegarde/restauration.

Il n'est pas rare de tomber sur des imbrications de sauvegarde/restauration à plusieurs niveaux sans nécessité.

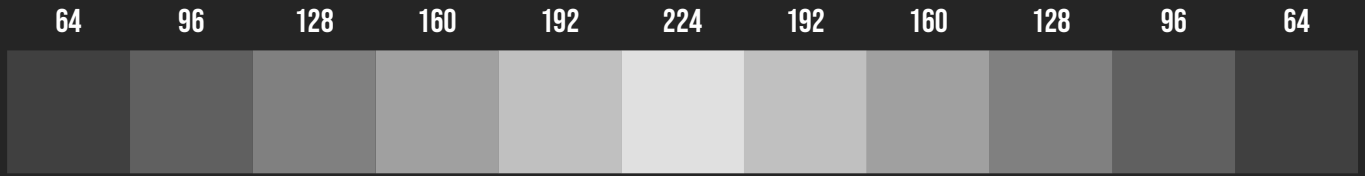




## AMÉLIORER L'EFFICACITÉ DES FILTRES

97/111

Un dégradé a peu de redondance de données  
Une compression par dictionnaire sera inefficace



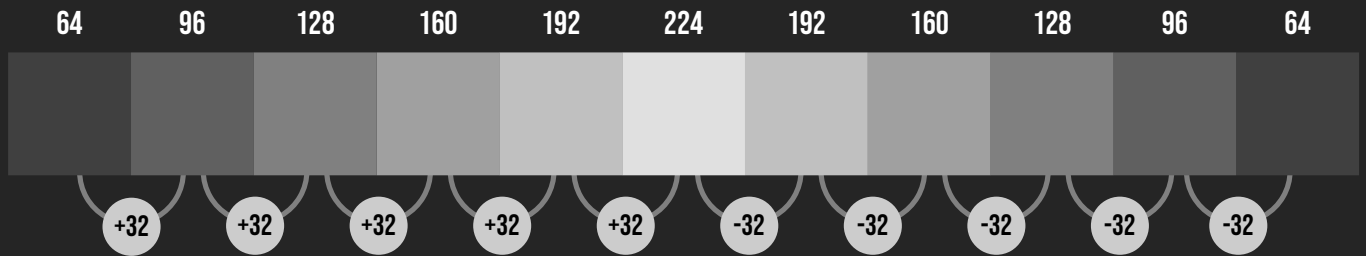
**Faible taux de compression**

Pour comprendre le fonctionnement des prédicteurs, prenons l'exemple d'un dégradé de gris.

Celui-ci ne présente pas de suites de valeurs redondantes rendant une compression par dictionnaire (comme Flate) peu efficace.

**98/111**

Le prédicteur Sub calcule la différence entre chaque pixel



**Prédicteur Sub**

En regardant de plus près, on peut cependant faire ressortir un motif.

Du premier au deuxième pixel, on a fait +32 et, cela, jusqu'au sixième pixel à partir duquel on a fait -32 (c'est le principe des dérivées en mathématiques).

Si les valeurs de pixels du dégradé ne sont pas compressibles, les différences entre chaque pixel, elles, le sont !

Tenir compte de la valeur du pixel précédent, c'est le job du prédicteur « Sub ».

Des répétitions apparaissent après application du prédicteur  
Seule la première valeur est conservée telle qu'elle

64	32	32	32	32	32	-32	-32	-32	-32	-32
----	----	----	----	----	----	-----	-----	-----	-----	-----

**Fort taux de compression**

Des répétitions apparaissent après application du prédicteur. Seule la première valeur ne bouge pas dans notre cas.

Faire apparaître autant de répétitions aura un impact favorable sur le taux de compression.

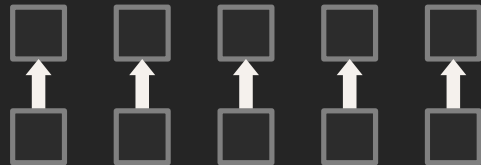
**100/111**

## Prédicteurs disponibles (1/2)

- Sub



- Up



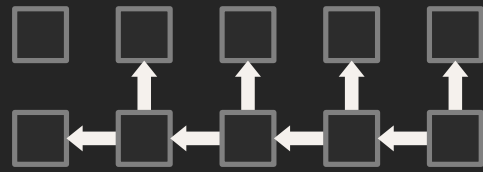
D'autres prédicteurs sont disponibles qui prennent en compte la nature bidimensionnelle des images.

On vient de voir le prédicteur «Sub», il y a aussi le prédicteur «Up» qui fait la différence avec les pixels du dessus.

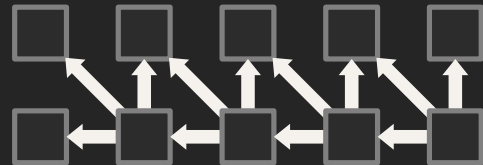
# 101/111

## Prédicteurs disponibles (2/2)

- Average



- Paeth

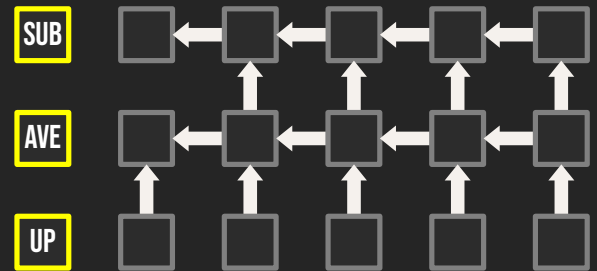


Le prédicteur « Average » fait la différence avec la moyenne des pixels au-dessus et à gauche.

Le prédicteur « Paeth » (du nom de son inventeur) prend en compte les pixels au-dessus à gauche, au-dessus et à gauche.

# Un prédicteur PNG par ligne

- 1 octet en début de ligne  
0 = aucun / 1 = sub / 2 = up / 3 = average / 4 = paeth
- Comment trouver la combinaison optimale?
  - Test de toutes les combinaisons possibles
  - Calcul d'entropie
  - Réseau neuronal?



Il existe deux types de prédicteurs : les prédicteurs TIFF et les prédicteurs PNG.

Les prédicteurs TIFF sont valables pour l'image toute entière.

Les prédicteurs PNG sont définis pour chaque ligne, permettant d'adapter le prédicteur en fonction du contenu de chaque ligne.

Les prédicteurs PNG présentent la difficulté du choix du meilleur prédicteur pour une ligne.



## UTILISER LE MEILLEUR FILTRE

Maintenant qu'on a fait le ménage sur les données inutiles au rendu d'un PDF, voyons s'il est possible de mieux encoder le fichier afin de gagner de la place.





## Histoire de filtres

- PDF 1.0 ( 1993 )
  - JPEG
  - CCITT Group 3/4
  - LZW
  - RLE
  - Ascii85
  - Hex
  - Prédicteur TIFF
- PDF 1.2 ( 1996 )
  - Flate
  - Prédicteurs PNG
- PDF 1.4 ( 2001 )
  - JBIG2
- PDF 1.5 ( 2003 )
  - JPEG 2000

Différents encodages se sont ajoutés au format PDF au fil du temps.

Le dernier filtre en date a été ajouté en 2003 au format PDF 1.5 : c'est le JPEG 2000.

Peu de logiciels y ont recourt et il n'est pas inclus dans les encodages autorisés pour le PDF/A.

# Remplacement de filtres

- Ascii85 → *Rien!*
  - Hex → *Rien!*
  - LZW → Flate
  - Flate → JPEG 2000?  
lossless
  - JPEG → JPEG 2000?  
lossy
  - CCITT → JBIG2
- Ascii85 et Hex
    - Pas de chiffrement!
    - Pour des PDF ASCII
    - Surpoids



Une optimisation simple des filtres consiste à remplacer les filtres par des équivalents plus performants.

Pour les filtres Ascii85 et Hex, on ne les remplace par rien du tout : de par leur conception, ils font grossir les données qu'ils encodent. Ils ont aujourd'hui un intérêt avant tout pédagogique.

LZW est un algorithme beaucoup moins performant que Flate.

JPEG 2000 est aussi plus performant que le JPEG.

Dans les cas où Flate est utilisé pour la compression d'images, le mode sans perte de JPEG 2000 peut le remplacer.

# JPEG 2000 or not JPEG 2000 ?

- Réencodage JPEG  
cumul d'artéfacts, limites de gain
- JPEG 2000 lossless  
peu efficace pour les masques
- Limites d'utilisation  
autorisé à partir de PDF/A-2



2000

Même s'il est peu utilisé aujourd'hui, le JPEG 2000 est largement supporté (Acrobat Reader, PDF.js, Visionneuse Gnome...).

Comme pour la plupart des compressions avec perte, le JPEG 2000 est un format intéressant dès l'instant où il est le format originel d'une image. Fonctionnant sur un algorithme très différent du JPEG classique, les artéfacts qu'il génère s'accumuleront avec ceux produits par ce dernier.

Il supporte également un mode de compression sans perte mais celui-ci reste souvent moins efficace que l'algorithme Deflate couplé à des prédictors en ce qui concerne les masques.

Enfin, il n'est pas autorisé dans les PDF/A...



# Optimisation du JPEG

- **Progressive vs baseline**  
2 à 10 % de gain en général
- **Codage arithmétique**  
5 à 10 % de gain, mais non supporté
- **Vignette**  
inutilisée par le PDF
- **CMJN → RVB**  
25 à 30 % de gain, mais perte de fonctionnalité
- **Tables de quantisation et Huffman**  
jusqu'à 30 % de gain
- **Taux de compression**  
gain mais perte d'information
- **Commentaires inutiles**
- **JpegTran, MozJPEG...**

La très large adoption du JPEG depuis son apparition dans les années 90 a entraîné l'apparition de nombreuses astuces afin d'optimiser son poids.

# Optimisation de Flate

- Utilisation de Zopfli

- 👍 Meilleur **taux** de compression que Zlib
- 👍 Les données compressées par Zopfli restent **lisibles** par Zlib
- 👎 **Temps** de compression très important



Le taux de compression du filtre Flate peut être amélioré en utilisant l'algorithme Zopfli.

Cela permet de rester compatible avec les décompresseurs Flate standard.

En revanche, le gain est obtenu au détriment du temps de compression.



## REGROUPER POUR MIEUX COMPRESSER

Pour améliorer le taux de compression, on peut recourir à la technique du regroupement.

110/111



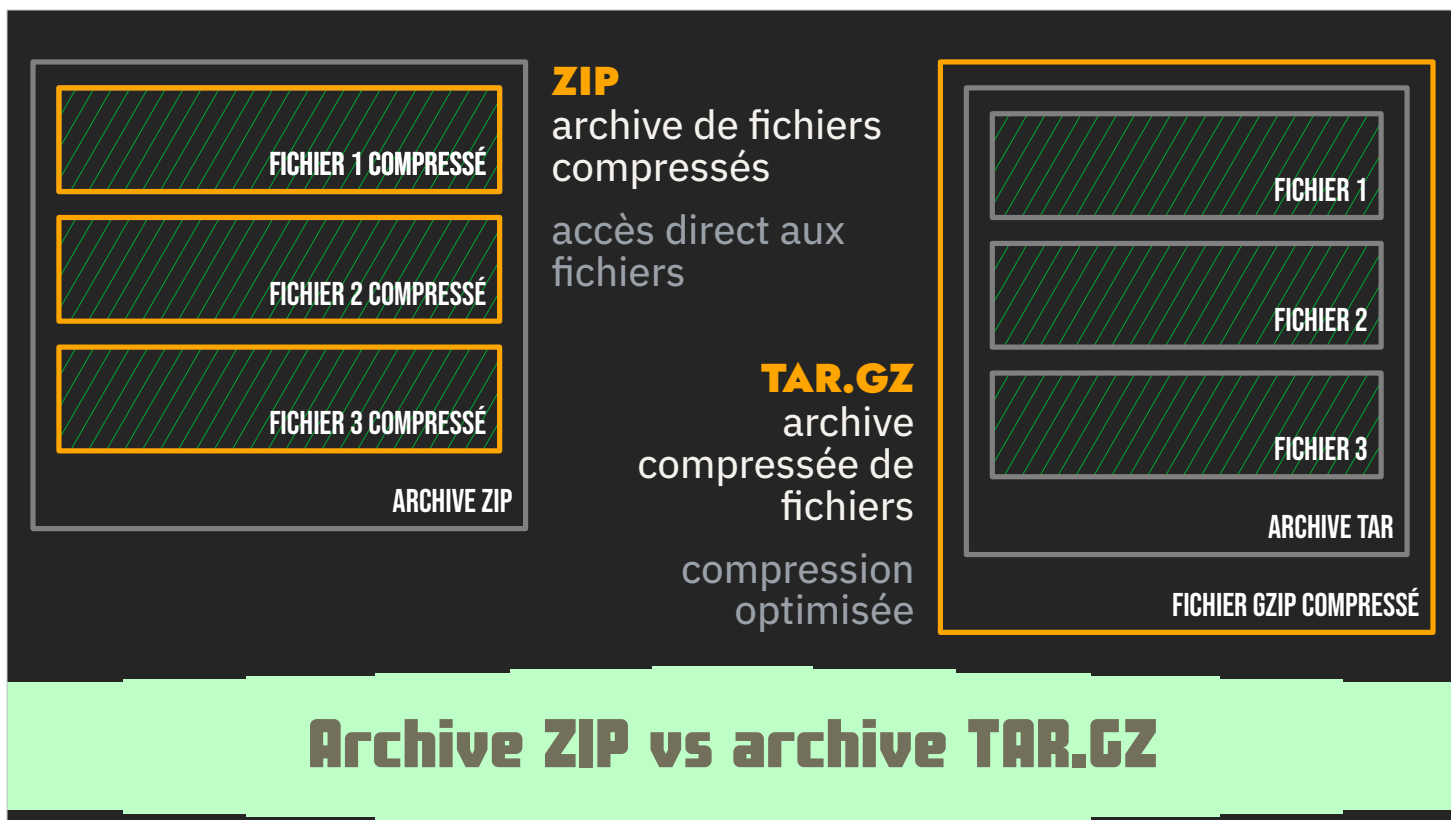
# Une fonctionnalité introduite en 2003

- Avant PDF 1.5
  - Seuls les flux peuvent être compressés
  - Références croisées et objets sans flux restent en clair
- À partir de PDF 1.5
  - On peut regrouper références croisées et objets sans flux dans le flux d'un objet
  - ... et utiliser les prédicteurs !
  - ... et les compresser !

Avant PDF 1.5, seuls les flux pouvaient être compressés. Les références croisées et les objets sans flux restaient en clair.

À partir de PDF 1.5, il est possible de regrouper les références croisées et les objets sans flux dans le flux d'un objet.

Cela permet d'utiliser les prédicteurs et de les compresser. Avec des fichiers PDF prenant de plus en plus de poids, le nombre de références croisées et d'objets sans flux représentent une part non négligeable du poids total du fichier.



En regroupant plusieurs objets et en les compressant d'un seul tenant, cela permet d'optimiser la compression à l'instar de ce que font les archives .tar.gz.

Alors que les algorithmes de compression de Gzip et Zip restent proches, le fait de regrouper les données à compresser permet d'augmenter le taux de compression.





## CUMULER LES FILTRES



PDF a aussi la particularité de pouvoir cumuler les encodages.

113/111

# Combinaisons intéressantes

- Certaines combinaisons peuvent être gagnantes
  - 👍 image-brute → RLE → flate → image-compressée
  - 👍 image-brute → JPEG-baseline → flate → image-compressée
- Quand certaines sont interdites
  - 🚫 image-brute → JPEG-progressif → flate → ❌



Certaines combinaisons peuvent être gagnantes en fonction des données à compresser.

Par exemple, si on utilise l'algorithme RLE et ensuite l'algorithme Flate, on peut voir des gains apparaître.

C'est également valable si on tente de compresser des images JPEG.

Le seul bémol est de suffisamment faire attention aux combinaisons employées car certaines ne sont pas autorisées par le format PDF.

# Des prédicteurs pour RLE ?

- Problèmes

- Les prédicteurs sont des paramètres, pas des filtres
- Ils sont réservés à LZW et Flate
- Ils sont effectifs sur des données à 2 dimensions

- Solution

- Flate dispose d'un mode « NoCompression »



L'algorithme RLE est un algorithme très simple de compression qui se concentre sur les suites d'octets identiques.

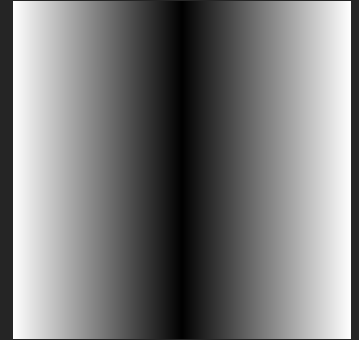
Si ses performances n'ont rien de comparables aux algorithmes de compression moderne, il n'en reste pas moins qu'il compresse admirablement bien les suites d'octets identiques, bien mieux que ne saurait le faire Flate par exemple.

Si seulement on pouvait profiter des prédicteurs pour faire apparaître des suites d'octets identiques à l'algorithme RLE... Malheureusement les prédicteurs sont réservés à LZW et Flate.

Et si on utilisait une particularité de Flate : le mode « NoCompression » ?

## Exemple de prédicteurs avec RLE

- Image test : 512×512 pixels, dégradé blanc → noir → blanc
- Comparatif des poids
  - Brut = 262 144 octets
  - Prédicteur + RLE = 5 703 octets
  - Flate = 1 765 octets
  - Prédicteur + Flate = 877 octets
  - Prédicteur + RLE + Flate = 134 octets
  - WebP = 654 octets
  - JPEG 2000 = 513 octets
  - JPEG XL = 414 octets



Prenons une image de 512 × 512 pixels en niveau de gris sur laquelle on a dessiné un dégradé du blanc au noir puis au blanc.

Brut, l'image pèse 256 Ko.\*

En utilisant les prédicteurs avec RLE, on tombe à 5703 octets.

Ce n'est pas encore folichon étant donné que Flate (Zopfli) fait mieux avec 1765 octets, et les prédicteurs avec Flate permettent même de descendre à 877 octets.

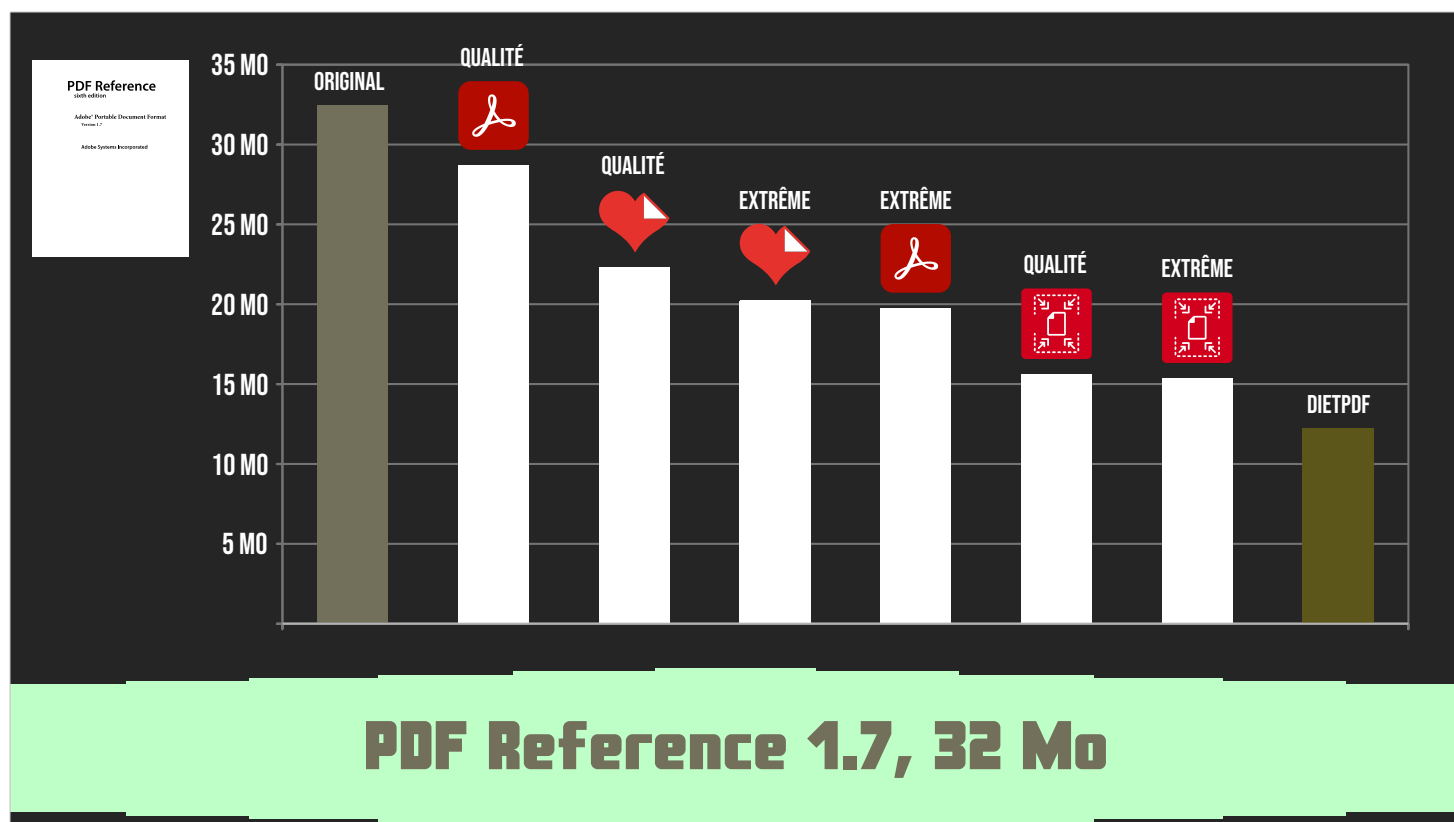
Mais si on utilise les prédicteurs, puis RLE, puis Flate, on descend à un très honorable 134 octets. À titre de comparaison, c'est mieux que le WebP ou le JPEG XL (qui ne sont pas supportés par le PDF).



**Et DietPDF ?**

Et DietPDF dans tout ça ?

**117/111**



Voici un test de différents compresseurs et réglages sur le fichier « PDF Reference » (version 1.7).

Le fichier original fait 32 Mo.

Adobe Acrobat online et iLovePDF descendent aux alentours de 20 Mo.

AVEPDF fait mieux en descendant aux alentours de 15 Mo.

DietPDF arrive aux alentours de 12 Mo.

Dans cet exemple choisi, donc biaisé, DietPDF s'en sort bien du fait de la nature du fichier PDF traité. Celui-ci contient beaucoup de structures et d'objets vectoriels, des types de contenu sur lesquels DietPDF opère plus d'optimisations que la plupart des compresseurs, hormis AVEPDF. La différence entre DietPDF et ce dernier tient dans l'utilisation de Zopfli.

# MERCI!

- Merci aux organisateurs et à Virginie Férey
- DietPDF : <https://github.com/Zigazou/dietpdf-haskell>
- Moi sur les internets
  - Github : <https://github.com/zigazou>
  - Twitter : @zigazou
  - Mail : [zigazou@protonmail.com](mailto:zigazou@protonmail.com)

Merci de votre attention !

Merci aux organisateurs.

Merci à Virginie Férey.

Vous pouvez me retrouver sur les internets aux endroits suivants.