



DIRECTION GÉNÉRALE DES FINANCES PUBLIQUES
SERVICE DES SYSTÈMES D'INFORMATION
SOUS-DIRECTION ETUDES ET DÉVELOPPEMENT
BUREAU SI-1A DMOD
4, AVENUE MONTAIGNE
93468 NOISY-LE-GRAND CEDEX

CHARTRE DE DÉVELOPPEMENT XML/XSL

RÉF. : CHARTREDevXMLXSL

VERSION : 0.4 DU 15/10/2008

STATUT : ~~APPLICABLE~~ / EN VALIDATION / ~~EN-COURS~~ / ~~SUPPORT~~

Objet et domaine d'application
<p>En conformité avec les axes stratégiques d'adoption de normes, de standards et d'utilisation de logiciels libres, la charte de développement XML/XSL regroupe des règles classées par catégorie en s'appuyant largement sur les critères communément admis pour l'évaluation de code. Cette charte fixe les exigences qualité attendues avec trois niveaux de contraintes.</p> <p>La grille d'évaluation reprend les règles de la charte et permet de mesurer l'écart entre le code livré et la charte dans le cadre de la recette technique ou dans le cadre d'audit de code.</p> <p>La charte et la grille s'appliquent à tout nouveau projet intégrant des développements en XML/XSL. Ces deux documents sont référencés dans le cahier des clauses techniques particulières puis imposés dans le plan qualité projet.</p> <p>Les projets existants intégreront ces règles au fur et à mesure des maintenances évolutives.</p>

Document de portée	Particulière : ne concerne que ses destinataires	
	Générale : application à tous - Programme Copernic et DGFIP	✓

		Projet/bascule ou applicatif en		
		Production	IRA / IRIA / ISIQ	Développement
Document d'application	Immédiate : les règles présentées s'appliquent immédiatement			
	Sur le flux : application immédiate sur les nouveaux développements			✓
	A planifier : renvoi d'un planning de mise en œuvre au bureau SI-1A			

Document d'importance	Vitale : sa mise en œuvre conditionne la cohérence du programme	✓
	Particulière : la mise en œuvre présente un intérêt particulier	
	Informative : solution d'un projet, présentant un intérêt général, par ex.	

PAGE DE GESTION

	Nom	Bureau	Date
Rédigé par	Romain PELISSE (Atos Origin) Lionel DURIEZ (Atos Origin) Malik SAHEB (Red Hat)	SI-1A/DMOD	30/09/2008
Approuvé par	Françoise PAYEN Xavier CHATELAIN Maurizio DE CECCO	SI-1A/DMOD	04/11/2008
Validé par	Eric PRIOL Nicolas BOITARD	SI-1A/DMOD	

Historique des versions				
Version	Date	Site/Rubrique	Rédigé par	Commentaires / modifications apportées
0.1	17/06/2008	http://publicop.eole.dgi/doc32045	Romain PELISSE	Document initial
0.2	20/08/09	http://publicop.eole.dgi/doc32045	Romain PELISSE Malik SAHEB	Document d'entrée à la deuxième réunion du groupe de travail, ajout de la partie XSL, intégration des remarques de la première réunion du groupe de travail.
0.3	26/09/2008	http://publicop.eole.dgi/doc32045	Romain PELISSE Malik SAHEB	Document d'entrée à la troisième réunion du groupe de travail, intégration des remarques de la précédente réunion du groupe de travail.
0.4	15/10/2008	http://publicop.eole.dgi/doc32045	Romain PELISSE Lionel DURIEZ	Relecture et finalisation.

Grille de lecture

Selon le profil du lecteur, la revue complète du présent document n'est pas forcément nécessaire. En effet :

- **le chef de projet informatique de la MOE** pourra limiter la lecture du présent document aux chapitres introductifs (1 à 2), et aux tableaux récapitulatifs des règles, en portant une attention particulière aux contraintes associées. A noter que la grille d'évaluation est un outil d'aide à l'évaluation de la conformité du code par rapport à la charte et constitue une aide au pilotage du projet ;
- **l'architecte** devra prendre connaissance de l'ensemble du document, mais portera particulièrement son attention sur les chapitres 3, 4, 5 et 7 (et le chapitre 6 si l'utilisation de XSL est dans le périmètre du projet). Il veillera notamment à faciliter le respect de l'ensemble des règles. Les besoins non couverts par la charte de développement XML/XSL feront l'objet d'une demande justifiée transmise au bureau SI-1A/DMOD (cf. note sur le processus de mise à jour <http://publicop.eole.dgi/doc97>), et ce avant le CAI du projet ou avant (un à deux mois) le début des développements ;
- **le développeur** devra effectuer une lecture attentive de l'ensemble du document, et tout particulièrement des règles qui lui sont nouvelles, ainsi que leurs justifications. Les chapitres 3, 4, 5, 7 (et 6 s'il doit développer des feuilles XSL) doivent donc être étudiés avec attention ;
- **le valideur** pourra concentrer sa lecture sur les chapitres 1 et 2, et sur les tableaux récapitulatifs des règles dans les chapitres suivants. La grille d'évaluation lui sera utile pour formaliser le résultat de l'audit de code et pour fournir un état des lieux du respect de la charte.

SOMMAIRE

1	<u>Présentation de la charte de développement XML/XSL</u>	7
1.1	<u>Contexte et objectifs stratégiques sous-jacents</u>	7
1.2	<u>Le périmètre de la charte</u>	7
1.3	<u>Les versions XML/XSL supportées</u>	8
1.4	<u>Les métiers ciblés</u>	10
1.5	<u>Application de la charte</u>	10
1.6	<u>Processus d'évaluation ou d'audit</u>	10
1.7	<u>Gestions des amendements et des évolutions</u>	10
2	<u>Démarche et conventions adoptées</u>	11
2.1	<u>Sources</u>	11
2.2	<u>Structure</u>	11
2.3	<u>Critères de sélection des règles</u>	12
2.4	<u>Conventions</u>	13
3	<u>Règles sur le langage de balisage XML</u>	15
3.1	<u>Introduction</u>	15
3.2	<u>Prologue du document</u>	15
3.3	<u>L'encodage du document</u>	17
3.4	<u>Instruction de traitement et entités</u>	20
3.5	<u>Utilisation des attributs spéciaux (xsi: et xml:)</u>	21
3.6	<u>Nommage</u>	23
3.7	<u>Documentation</u>	26
3.8	<u>Mise en forme</u>	27
3.9	<u>Conception de document XML</u>	33
3.10	<u>Utilisation des patterns et anti-patterns XML</u>	42
3.11	<u>Performance</u>	47
3.12	<u>Génération et publication de flux XML</u>	48
4	<u>Règles d'utilisation des espaces de noms XML</u>	51
4.1	<u>A propos de XML Namespace</u>	51
4.2	<u>Conventions de nommage</u>	51
5	<u>Règles associées à la validation XSD</u>	52
5.1	<u>Finalité, critères de sélection, sources et contraintes</u>	52
5.2	<u>Documents XML et XSD</u>	53
5.3	<u>Un document XSD est avant tout un document XML</u>	53
5.4	<u>Structuration du document</u>	55

5.5	<u>Règles relatives aux Namespaces au sein des XSD</u>	57
5.6	<u>Règles de définition de type</u>	61
5.7	<u>Règles issues des « patterns » XML</u>	65
5.8	<u>Règles de gestion des commentaires</u>	66
5.9	<u>Règles sur l'utilisation des PSVI</u>	67
6	<u>XSL</u>	68
6.1	<u>Règles relatives à XPath</u>	68
6.2	<u>Règles relatives à XSLT</u>	71
6.3	<u>Règles relatives à XSL-FO</u>	105
7	<u>Règles relatives à la sécurité</u>	106
7.1	<u>Finalité, critères de sélection, sources et contraintes</u>	106
7.2	<u>Règles déjà citées et ayant un impact positif sur la sécurité</u>	106
7.3	<u>Règles spécifiques à la sécurité</u>	108
8	<u>Annexes</u>	109
8.1	<u>Bibliographie</u>	109
8.2	<u>Lexique</u>	112
8.3	<u>Sommaire détaillé</u>	114
8.4	<u>Licence du présent document</u>	122

1 Présentation de la charte de développement XML/XSL

1.1 Contexte et objectifs stratégiques sous-jacents

La réalisation du système d'information fiscal de la DGFIP doit répondre aux objectifs majeurs suivants :

- maîtrise du système d'information ;
- pérennité du système d'information ;
- indépendance par rapport aux technologies et aux fournisseurs.

Il s'ensuit une stratégie informatique visant, notamment, à :

- respecter les normes et standards du marché ;
- normaliser le système d'information ;
- favoriser la maintenabilité et l'exploitabilité des composants ;
- imposer le respect des normes et standards du service des systèmes d'information de la DGFIP mais aussi aux entreprises intervenant au titre de la sous-traitance.

La charte de développement XML/XSL s'inscrit dans cette stratégie. Sa finalité est bien d'unifier, faciliter et promouvoir la qualité des fichiers XML, XSD et XSL tout au long du cycle de vie des applicatifs, afin de valider les livrables et faciliter la maintenance, c'est-à-dire :

- améliorer la lisibilité des fichiers ;
- faciliter la maintenabilité des fichiers XML, XSD, comme XSL ;
- uniformiser au maximum la syntaxe utilisée.

La charte n'est pas un guide de développement et ne présente pas « comment développer en XML/XSL ». Elle s'efforce de ne retenir que les règles utiles et pertinentes, et non les habitudes de développement.

1.2 Le périmètre de la charte

L'objectif de cette charte est de standardiser les fichiers XML « non générés » qui seront :

- maintenus ou utilisés de manière manuelle (fichiers de configuration) ;
- utilisés au cours des échanges de données (par exemple : entre MAP et MAS) ;
- exposés à l'extérieur du système d'information de la DGFIP.

En outre, une partie de cette charte n'est bien évidemment pas applicable directement aux documents XML générés. Il s'agit principalement des règles de présentation et de lisibilité qui ne peuvent pas être exigées d'un projet qui n'a pas le contrôle de la mise en forme d'un document XML. Le respect des conventions de cette charte est un critère important dans le choix des générateurs choisis et recommandés par le bureau SI-1A.

1.3 Les versions XML/XSL supportées

La présente charte concerne les versions autorisées dans la matrice technologique ^[2b].

1.3.1 Les différentes normes ou recommandations XML


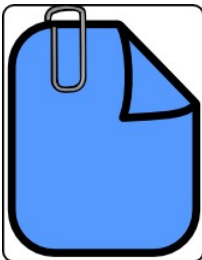

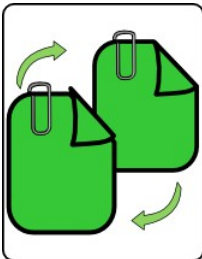


<i>Spécification</i>	<i>Version</i>
XML	1.0
XML Namespace	1.0
XML Schema (XSD) Structures Datatypes Component Designators	1.0

1.3.2 Les différentes normes ou recommandations XSL

<i>Spécification</i>	<i>Version</i>
XSL	1.0
XPath	1
XSLT	1
XSLT-FO	1.1

1.3.3 Les usages

Selon l'usage fait de XML, le périmètre d'application de la charte n'est pas le même. Pour chacune des règles, le tableau de synthèse indiquera l'applicabilité de la règle.

<i>Usage</i>	<i>Symboles</i>
Fichiers de configuration hors du périmètre projet (exemple : JBoss, fichiers du SPPC de ex SI2)	
Fichiers de configuration spécifiés par le projet (exemple : configuration d'un cache interne)	
Fichiers générés par des librairies tierces (exemple : fichiers WSDL, Castor, XMLBeans)	
Fichiers d'échange spécifiés par le projet ou un autre projet DGFIP	
Fichiers d'échange spécifiés hors DGFIP (exemple : fichier INSEE)	
Fichiers d'échange spécifié par la DGFIP exposé à l'extérieur	

1.4 Les métiers ciblés

La charte XML/XSL concerne les maîtrises d'œuvre internes ou externes (sociétés de services informatiques) et cible plus particulièrement les profils suivants :

- le chef de projet informatique de la MOE ;
- l'architecte ;
- le développeur ;
- l'analyste-recetteur.

1.5 Application de la charte

La charte de développement XML/XSL s'applique à tout nouveau projet utilisant les technologies XML et XSL dans son développement, et tout nouveau *framework* générant ou utilisant du XML.

Les projets existants intégreront ces règles au fur et à mesure des maintenances évolutives, en se concentrant essentiellement sur les fichiers d'échanges et de configuration.

1.6 Processus d'évaluation ou d'audit

La grille d'évaluation reprend les règles de la charte et permet de mesurer l'écart entre le code livré et la charte dans le cadre de la recette technique ou dans le cadre d'audit de code.

1.7 Gestions des amendements et des évolutions

Les demandes d'amendement ou d'évolution seront transmises dans la Balf du bureau SI-1A/DMOD avec comme objet « [pour attribution] charte de développement XML/XSL ».

Les besoins non couverts par la charte XML/XSL feront l'objet d'une demande justifiée (cf. note sur le processus de mise à jour <http://publicop.eole.dgi/doc97>), et ce, avant le CAI du projet ou avant le début des développements.

Les demandes d'évolution ou les anomalies seront directement saisies dans la base de bogues du projet « Norme SODA, composant charte XML/XSL »,

(http://venezia.dev.impots/plugins/bugzilla/?group_id=18).

Les éventuelles modifications de la Charte de Développement XML/XSL seront étudiées par le Comité de GESTION de l'Atelier SODA (COGESTA) et le Comité d'Architecture des Chefs de Projet (CACP), puis présentées au Comité de PILOTAGE (COPIL) SODA.

2 Démarche et conventions adoptées

2.1 Sources

La présente charte s'appuie principalement sur les spécifications suivantes du W3C :

- la spécification XML 1.0 (4ième édition), datant du 16/08/2006, et disponible à l'adresse <http://www.w3.org/TR/2006/REC-xml-20060816/> ^[3];
- la spécification XML Schema, datant du 02/15/2001, et disponible à l'adresse <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/> ^[3c];
- la spécification XSLT 1.0, datant du 16/11/1999, et disponible à l'adresse <http://www.w3.org/TR/xslt> ^[3e].

Cette charte s'appuie également sur un certain nombre de documents internes à la DGFIP , dont :

- des notes de la DASP :
 - « Normes et standards HTML dans la conception d'applicatif » ^[2h] ;
 - « Synthèse UTF8 » ^[2i] ;
 - « Norme d'encodage UTF8 » ^[2j] ;
- un document de l'ADAE, « *Le Guide UML-XML* » ^[1e].

2.2 Structure

Le présent document regroupe les règles en grands domaines :

- règles d'organisation d'un fichier XML, elles mêmes découpées en domaines :
 - prologue du document ;
 - règles sur l'encodage ;
 - règles d'utilisation des instructions de traitement et des entités ;
 - règles d'utilisation des attributs spéciaux ;
 - règles de nommage ;
 - règles de documentation ;
 - règles de mise en forme ;
 - règles de conception ;
 - règles d'utilisation des *patterns* ;
 - règles pour la performance ;
 - règles sur la génération et publication de document ;
- règles associées à l'utilisation des espace de noms ;
- règles d'organisation d'un fichier XSD, elles mêmes découpées en domaines :
 - règles d'utilisation des espaces de noms au sein des XSD ;
 - règles de définition de type ;

- règles issues de patterns XML ;
- règles de gestion des commentaires ;
- règles d'utilisation des PSVI ;
- règles d'organisation et de codage d'un fichier XSL, elles mêmes découpées en domaines ;
 - règles d'utilisation de Xpath ;
 - règles relatives à XSLT ;
 - règles d'utilisation de XSL-FO ;
- règles de sécurité dans les technologies XML, elles mêmes découpées en domaines :
 - règles déjà citées et ayant un impact positif sur la sécurité ;
 - règles spécifiques à la sécurité.

Chaque groupe de règles listé ci-dessus respecte le plan suivant :

- finalité, critères de sélection, sources et contraintes ;
- détail des règles ;
- éventuels exemples et/ou exceptions.

2.3 Critères de sélection des règles

L'introduction d'une nouvelle règle est appréciée par rapport à une liste définie de critères caractérisant la Charte de Développement XML/XSL.

L'acceptation d'une règle dépend du degré de satisfaction des critères.

Ces critères sont les suivants :

- la familiarité :
 - acceptation / utilisation par la communauté (interne ou externe) ;
- la complexité :
 - outillage (facilité de mise en oeuvre avec des outils, degré d'automatisation) ;
 - clarté (compréhension) ;
- la cohérence :
 - interne des règles ;
 - par rapport au système d'information de la DGFIP ;
 - par rapport aux normes et standards en vigueur ;
- les effets positifs de la règle sur la qualité du code ou de l'architecture, les avantages de la mise en place de cette règle sur l'ingénierie logicielle et plus particulièrement en termes de :
 - bonne et rapide compréhension ;
 - lisibilité du code source ;

- uniformité du code source ;
- maintenabilité, évolutivité ;
- portabilité, compatibilité, scalabilité ;
- modularité, réutilisabilité ;
- sécurité ;
- performance.


2.4 Conventions

Les qualifications et sigles employés sont les suivants :

2.4.1 Nature des règles

<i>Qualification</i>	<i>Sigle</i>	<i>Explication</i>
Norme	NOR	« Une norme est un référentiel publié par un organisme de normalisation comme ECMA, W3C ou ISO. Les organismes de normalisation sont des organismes reconnus au niveau national ou international. Ils peuvent être constitués soit par des États, soit par des consortiums internationaux de professionnels. Comme la langue anglaise ne marque pas la différence entre norme et standard (« norme » se dit « <i>standard</i> »), on parle pour les normes de standards <i>de jure</i> . » (Wikipédia)
Standard de fait	STD	« Un standard est un référentiel publié par une autre entité. En fait on ne parle de standard qu'à partir du moment où le référentiel a une diffusion large, on parle alors de standard <i>de facto</i> (standard de fait), en informatique les formats PDF ou les fichiers Microsoft Word en sont des exemples très connus. » (Wikipédia)
Standard DGFIP	STD-DGFIP	Standard reconnu au sein des projets du système d'information de la DGFIP

2.4.2 Contraintes d'utilisation

<i>Qualification</i>	<i>Sigle</i>	<i>Explication</i>
<u>Obligatoire</u>		Le non respect de la règle justifie la non conformité ou l'émission de réserves sur un livrable.
<u>Fortement recommandé</u>		Obligatoire, mais peut être soumise à une dérogation exceptionnelle sur la base de motivations.
<u>Recommandé</u>		Non obligatoire, mais est retenue dans le calcul d'un indicateur de qualité. En-dessous du seuil de qualité, un livrable peut être déclaré non conforme ou avec réserves.

La typographie suivante est utilisée pour rappeler les contraintes d'utilisation à travers le document :

- en souligné gras pour la notion d' **Obligatoire** ;
- en gras pour la notion de **Fortement recommandé** ;
- en souligné pour la notion de Recommandé.

3 Règles sur le langage de balisage XML




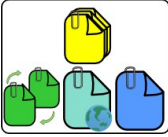

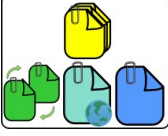

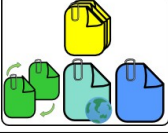

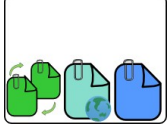

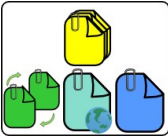
3.1 Introduction

Les règles décrites dans ce chapitre ne concernent pas directement le flux ou le fichier XML en tant que tel, mais traitent de l'utilisation et des bonnes pratiques qui entourent sa fabrication et la diffusion des flux et des fichiers XML.

3.2 Prologue du document

3.2.1 Finalité, critères de sélection, sources et contraintes

Ces règles regroupent les recommandations relatives au prologue d'un document XML, ainsi qu'aux pré-requis nécessaires à son analyse par un parseur XML pour que le document soit bien formé.

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[PROLOGUE -XmlValid]	Le document doit être conforme au format XML 1.0.	• W3C	NOR		
[PROLOGUE -Entete]	Le document XML doit contenir un entête.	• W3C	NOR		
[PROLOGUE -XmlVersion]	Le document doit utiliser la version 1.0 de la spécification XML, de manière explicite.	• DMOD	STD-DGFIP		
[PROLOGUE -encodage]	Le document doit utiliser l'encodage UTF-8 de manière explicite.	• DASP - synthèse UTF-8 ^[2]	STD-DGFIP		
[PROLOGUE -Standalone]	L'utilisation de la déclaration standalone="yes" est recommandée dans le document.	• Effective XML ^[4] • ADAE - Guide UML-XML ^[1c]	STD		
[XML-EnteteFichier]	Un document XML doit contenir un entête de fichier avec les informations minimales sur son propos.	• DMOD	STD-DGFIP		

3.2.2 [PROLOGUE-XmlValid] Le document **doit** être conforme au format XML 1.0.

Les contraintes pour produire du XML « bien formé » sont bien évidemment à respecter. En aucun cas, un applicatif ne **doit** utiliser, générer ou exposer du XML mal formé.

3.2.3 [PROLOGUE-Entete] Le document XML **doit** contenir un entête.

En plus d'être bien formé, un document XML doit exposer un prologue correct qui respecte les règles décrites plus loin, relatives à la version de XML ([PROLOGUE-XmlVersion]) utilisée et à l'encodage ([PROLOGUE-encodage]).

3.2.4 [PROLOGUE-XmlVersion] Le document **doit** utiliser la version 1.0 de la spécification XML, de manière explicite.

La version retenue de la spécification XML est la 1.0 car elle est, aujourd'hui la plus utilisée.

Pour information, les évolutions éventuelles apportées par la version 1.1 concernent :

- le support de nouveaux langages s'appuie sur une version plus récente d'unicode (version 4.0) et donc davantage de langages (comme le Khmer) ;
- le caractère de retour de ligne IBM, NEL (0x85), a été normalisé comme un retour à la ligne. Cet ajout facilite le traitement de documents XML sur certains *mainframe* IBM utilisant ce caractère comme retour à la ligne.

Ainsi, l'ajout de nouveaux langages n'est pas un point important dans le contexte des projets DGFIP. En outre, certains parsers XML ne supportent pas la version 1.1 de la spécification XML.

3.2.5 [PROLOGUE-encodage] Le document **doit** utiliser l'encodage UTF-8 de manière explicite.

Tout document XML produit doit utiliser l'encodage UTF-8 et doit donc le spécifier dans son prologue.

3.2.6 [PROLOGUE-Standalone] L'utilisation de la déclaration *standalone="yes"* est recommandée dans le document.

La spécification décrit ainsi la déclaration *standalone* : « Dans une déclaration de document autonome, la valeur « *yes* » indique qu'il n'y a pas de déclaration de balisage externe à l'entité document (dans le sous-ensemble externe de DTD, ou dans une entité paramètre externe appelée dans le sous-ensemble interne) qui affecterait l'information transmise du processeur XML à l'application. »

Un document n'est pas autonome si des déclarations de balisage externes contiennent les déclarations suivantes :

- les déclarations d'attributs avec des valeurs implicites, si les élément auxquels s'appliquent ces attributs apparaissent dans le document sans spécification de valeur pour ces attributs, ou
- les déclarations d'entités externes (autres que *amp*, *lt*, *gt*, *apos* et *quot*), si des appels à ces entités apparaissent dans le document, ou
- les déclarations des attributs avec des valeurs sujettes à normalisation, où l'attribut apparaît dans le document avec une valeur qui va changer en raison de la normalisation.

La déclaration « `standalone="yes"` » interdit donc la modification du document XML par sa DTD. Le principal apport d'une DTD à un document XML est d'ajouter un ensemble de valeurs par défaut aux éléments et/ou attributs, mais ce mécanisme rend le document dépendant de sa DTD.

Dans le cadre des projets DGFIP, l'utilisation de DTD est proscrite ([XSD-PasDeDTD]), au profit de celle des XSD. En outre, il est préférable qu'un document XML ne s'appuie pas sur une DTD pour être complet. En effet, il est possible que le document XML soit utilisé sans que sa DTD soit appliquée, ou même simplement accessible au moment du traitement.

Pour toutes ces raisons, l'utilisation de la déclaration *standalone="yes"* est recommandée.

3.2.7 [XML-EnteteFichier] Un document XML doit contenir un entête de fichier avec les informations minimales sur son propos.

Tout fichier XML doit contenir une description globale de son rôle et des informations de maintenance nécessaires aux outils classiques de gestion de configuration (CVS,SVN,...) indiquant le nom de l'auteur, la date de modification ainsi qu'une description sur la mise à jour (« logs »).

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright : Ministère du budget, des comptes publics et de la fonction publique - France
  Description de fichier
  $Id$
  $Log$
  Createur: Prénom Nom
  Date de création: Date
-->
....
```

Note: Les valeurs telles que « `Id` » ou « `Log` » sont renseignées automatiquement par le serveur de gestion de configuration.



3.2.8 Exemple récapitulatif





L'exemple suivant respecte les règles [PROLOGUE-Entete], [PROLOGUE-encodage], [PROLOGUE-XmlValid], [PROLOGUE-XmlVersion] et [PROLOGUE-Standalone] décrites ci dessus.

Exemple

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

3.3 L'encodage du document

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[XML-ENC-FormeNormaleC]	Un document XML doit utiliser la Forme Normale C (« Normalization Form C »).	<ul style="list-style-type: none"> DASP - synthèse UTF-8^[2] 	STD-DGFIP		

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[XML-ENC-EntitesNecessaires]	On ne doit pas utiliser les entités pour les caractères accentués ou spéciaux à l'exception des caractères (<,"',>,&) interdits par XML.	• DMOD	STD-DGFIP		
[XML-GuillemetEtApostrophe]	On ne doit utiliser les entités ' et " pour représenter les apostrophes et les guillemets que si nécessaire.	• DMOD	STD-DGFIP		

3.3.1 [XML-ENC-FormeNormaleC] Un document XML **doit** utiliser la Forme Normale C (« *Normalization Form C* »).

L'unicode standard définit différentes manières équivalentes de représenter un caractère accentué ou spécial. La forme normale C a été retenue car elle assure la plus grande compatibilité possible. En outre, c'est la forme qui est recommandée pour la diffusion de documents sur Internet.

	Source	Forme Normal D	Forme Normal C
Symbole	Å	A + °	Å
Code point	00C5	0041 + 030A	00C5

3.3.2 [XML-ENC-EntitesNecessaires] On **ne doit pas** utiliser les entités pour les caractères accentués ou spéciaux à l'exception des caractères (<,"',>,&) interdits par XML.

L'encodage UTF-8, imposé pour l'ensemble des documents XML, supporte un large panel de caractères spéciaux. Ainsi, il n'est pas nécessaire d'utiliser des entités pour représenter des caractères spéciaux au sein d'un document XML.

La seule exception à cette règle est bien évidemment les 5 caractères utilisés pour définir la structure d'un document XML, soit : <,"',> et &. Ces caractères, pour pouvoir figurer dans un document XML, devront être représentés par leurs entités.

Caractère	Entité
&	&
<	<
>	>
'	'
"	"

Tableau 1: Liste des entités autorisées

En fait, seules ces entités sont prédéfinies au sein d'un document XML. Néanmoins, il est courant de voir apparaître des entités prédéfinies en HTML dans des documents XML, soit par méconnaissance, soit par amalgame avec le HTML.

Le contre-exemple ci dessous illustre l'utilisation d'entités pour représenter des caractères spéciaux :

Exemple	Contre exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <operations> <operation> 34 &gt; X + 3 </operation> </operations></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <operations> <operation> 34 > X + 3 </operation> </operations></pre>
<pre><?xml version="1.0" encoding="UTF-8"?> <donnee> <valeur>L'utilisation de caractères accentués ne nécessite pas d'utiliser des entités HTML ! </valeur> </donnee></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <donnee> <valeur>L'utilisation de caract&egrave;res accentu&eacute;s ne n&eacute;cessite pas d'utiliser des entités HTML ! </valeur> </donnee></pre>

3.3.3 [XML-GuillemetEtApostrophe] On ne **doit** utiliser les entités ' et " pour représenter les apostrophes et les guillemets que si nécessaire.

Les entités « ' » et « " » permettent de représenter respectivement les caractères « ' » et « " ». Ces entités permettent donc de représenter ces caractères dans des contextes où ils interféreraient avec la syntaxe XML.

Exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <!-- La valeur de l'attribut ci dessous est: Cette valeur contient des "'", les entités permettent donc de la représenter ainsi: --> <racine attribut='Cette valeur contient des &quot;&apos;&quot;.'/></pre>

S'il n'y a pas d'ambiguïté avec la syntaxe XML, il n'est pas utile d'utiliser ces entités :

Exemple	Contre exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <racine> L'utilisation "d'apostrophes" ne nécessite pas d'entité dans ce contexte. </racine></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <racine> L&apos;utilisation &quot;d&apos;apostrophes&quot; ne nécessite pas d'entité dans ce contexte. </racine></pre>

3.4 Instruction de traitement et entités

3.4.1 Finalités, critères de sélection, sources et contraintes




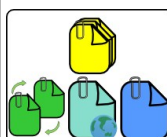
La spécification XML permet de placer, au sein du document XML, des informations à transmettre à l'applicatif traitant le document. Ces informations sont transmises à l'aide de la forme suivante :

Exemple
<pre><? cible instructions ?></pre>

L'entête de tout document XML ressemble en fait à une instruction de traitement:

Exemple
<pre><?xml version="1.0" encoding="" ?></pre>

Remarque: l'entête ressemble à une instruction de traitement mais n'en n'est pas une. En effet, la spécification interdit à une instruction de traitement de contenir, dans la chaîne de caractères 'cible' la série de caractères 'xml'.

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[XML-InstructionDeTraitementInterdit]	On ne doit pas utiliser les instructions de traitement au sein d'un document XML.	• DMOD	STD-DGFIP		
[Entite-NePasUtiliserEntites]	Pour conserver le document XML autonome, il est recommandé de ne pas utiliser d'entités (à l'exception des entités « & » , « < » , « > » , « ' » et « " »).	• DMOD	STD-DGFIP		

3.4.2 [XML-InstructionDeTraitementInterdit] On ne doit pas utiliser les instructions de traitement au sein d'un document XML.

Les instructions de traitement permettent de communiquer, à partir du document XML, des informations à l'applicatif qui traite le fichier. Ce mécanisme menace grandement l'indépendance du document XML, qui pourrait être analysé de manière très différente selon l'applicatif, et n'offre aucun avantage particulier au sein des projets DGFIP.

L'utilisation des instructions de traitement est donc proscrire.

Contre exemple

```
<racine>Texte du noeud</racine>
<?monPlugIn textType="french" maxColSize="10"?>
```

Cette interdiction fait écho à celle de la spécification SOAP qui interdit aussi explicitement l'utilisation d'instructions de traitement.


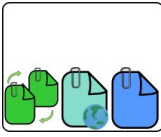


3.4.3 [Entite-NePasUtiliserEntites] Pour conserver le document XML autonome, il est recommandé de ne pas utiliser d'entités (à l'exception des entités « & » , « < » , « > » , « ' » et « " »).

Dans l'objectif de conserver les documents XML les plus autonomes possibles ([PROLOGUE-Standalone] , [CONCEP-PasDeBinaire] et [CONCEP-DocumentIndépendant]), l'utilisation des entités est déconseillée. L'utilisation d'entités, autre que celles admises par la spécification pour représenter les symboles utilisés par la syntaxe XML, rend le document dépendant de un ou plusieurs autres documents.

3.5 Utilisation des attributs spéciaux (xsi: et xml:)

3.5.1 Finalités, critères de sélection, sources et contraintes

Les spécifications du W3C définissent certains attributs spéciaux. Les règles d'utilisation de ces attributs tendent à normaliser leur utilisation au sein des projets DGFIP.

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[XML-NulliteElement]	Pour indiquer qu'un élément XML est nul, il est recommandé de ne pas utiliser l'attribut spécial « xsi:nil ».	<ul style="list-style-type: none"> Effective XML^[4] DMOD 	STD-DGFIP		
[XML-Espace]	Pour préserver les espaces au sein d'un élément, on doit utiliser l'attribut « xml:space ».	<ul style="list-style-type: none"> Effective XML^[4] DMOD 	STD-DGFIP		

3.5.2 [XML-NulliteElement] Pour indiquer qu'un élément XML est nul, il est recommandé de ne pas utiliser l'attribut spécial « xsi:nil ».

La spécification prévoit un attribut spécifique pour indiquer qu'un élément est nul (et non simplement vide). En effet, cette distinction peut parfois être nécessaire quand le document XML est, par exemple, issu d'une base de données relationnelle.

Néanmoins, l'usage de cet attribut n'est pas commun, et les cas d'utilisation où on fait la distinction entre « null » et vide sont rares. Il est donc recommandé, par souci de simplicité, de ne pas utiliser l'attribut « xsi:nil » pour indiquer qu'un élément est nul.

3.5.3 [XML-Espace] Pour préserver les espaces au sein d'un élément, on **doit** utiliser l'attribut « xml:space ».


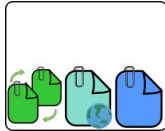



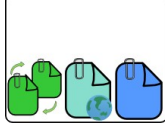



L'attribut « xml:space » permet d'indiquer au *framework* qui analyse le document XML que les espaces sont porteurs d'information et qu'il ne doit pas les transformer.

Exemple	Contre exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <poeme nom="Danse macabre" xml:space="preserve"> Fièvre, autant qu'un vivant, de sa noble stature, Avec son gros bouquet, son mouchoir et ses gants, Elle a la nonchalance et la désinvolture ... </poeme></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <poeme nom="Danse macabre"> Fièvre, autant qu'un vivant, de sa noble stature, Avec son gros bouquet, son mouchoir et ses gants, Elle a la nonchalance et la désinvolture ... </poeme></pre>

3.6 Nommage

3.6.1 Finalités, critères de sélection, sources et contraintes

L'ensemble des règles décrites dans ce chapitre porte sur la lisibilité des fichiers XML. Ces règles ne sont donc applicables qu'aux documents XML visant à être maintenus et/ou modifiés manuellement (fichier de configuration, de déploiement, de construction...).

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[NOM-Francais]	Les noms des éléments et des attributs doivent être écrits en français	<ul style="list-style-type: none"> • DMOD • Effective XML^[4] • ADAE - Guide UML-XML^[1c] 	STD-DGFIP		
[NOM-NomExplicite]	Les noms des éléments et des attributs doivent avoir des noms clairs et les plus explicites possible.	<ul style="list-style-type: none"> • W3C • DMOD • Effective XML^[4] • ADAE - Guide UML-XML^[1c] 	STD-DGFIP		
[NOM-CamelCase]	En commençant par une minuscule, les noms des éléments et des attributs doivent suivre la convention « camel Case », séparant les mots à l'aide de majuscules.	<ul style="list-style-type: none"> • W3C • DMOD • Effective XML^[4] • ADAE - Guide UML-XML^[1c] 	STD		
[NOM-PrefixInterdit]	Les noms des éléments et des attributs ne doivent pas être préfixés du caractère '_'.	<ul style="list-style-type: none"> • DMOD 	STD-DGFIP		
[NOM-UtilisationSymbole]	Les noms des éléments et des attributs ne doivent pas contenir de '_', ni de chiffre sauf si ces derniers contiennent une donnée métier.	<ul style="list-style-type: none"> • DMOD • ADAE - Guide UML-XML^[1c] 	STD-DGFIP		

3.6.2 [NOM-Francais] Les noms des éléments et des attributs doivent être écrits en français

L'utilisation de la langue française est **obligatoire**. Cette pratique permet une meilleure interprétation des données.

Pour des raisons de compatibilité et de portabilité, les caractères accentués sont interdits dans les noms des attributs et des balises. Ils sont néanmoins autorisés dans les commentaires XML.

Exemple	Contre exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <poeme nom="Danse macabre" xml:space="preserve"> Fière, autant qu'un vivant, de sa noble stature, Avec son gros bouquet, son mouchoir et ses gants, Elle a la nonchalance et la désinvolture ... </poeme></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <poem name="Danse macabre" xml:space="preserve"> Fière, autant qu'un vivant, de sa noble stature, Avec son gros bouquet, son mouchoir et ses gants, Elle a la nonchalance et la désinvolture ... </poem></pre>

3.6.3 [NOM-NomExplicite] Les noms des éléments et des attributs **doivent** avoir des noms clairs et les plus explicites possible.

Les noms des éléments et des attributs doivent être le plus auto descriptifs possible pour rester dans la logique de XML. Pour cela, Un nom d'élément ou d'attribut doit être, à minima, composé de 3 caractères et , au plus, de 36 caractères.

Exemple	Contre exemple
<pre><adresse> <destinataire>M.Dupont</destinataire> <lignes> <ligne>Résidence Leclerc</ligne> <ligne>5 avenue Général Leclerc</ligne> </lignes> <ville>Paris</ville> <codePostal>75104</codePostal> </adresse></pre>	<pre><adr> <dest>M.Dupont</dest> <lgs> <lg>Résidence Leclerc</lg> <lg>5 avenue Général Leclerc</lg> </lgs> <ville>Paris</ville> <code>75104</code> </adr></pre>

3.6.4 [NOM-CamelCase] En commençant par une *minuscule*, les noms des éléments et des attributs **doivent** suivre la convention « camel Case », séparant les mots à l'aide de majuscules.

La spécification XML est très souple dans la définition du nom des éléments et des attributs, et elle ne propose pas de bonne pratique ou de convention en ce domaine. En outre, le W3C n'a pas réussi à établir une convention unique à travers les différents langages normalisés de la sphère XML :

- XLST nomme ses balises intégralement en minuscules et sépare les mots clés par des « - » (exemple « xsl:for-each »);
- XSD utilise une convention camelCase et commence tous les mots par des minuscules (exemple « xsd:complexType »);
- DocBook utilise des minuscules et ne sépare pas les mots (exemples « chapterinfo », « biblioentry »);
- ...

Dans le cadre des projets DGFIP, c'est la syntaxe *camelCase* qui a été retenue, pour les raisons suivantes :

- elle évite l'utilisation de « _ » qui rend le document XML inutilement verbeux ;
- les francophones, comme les anglophones, reconnaissent les mots par leurs formes (et non en associant les différentes lettres), or, le *camelCase* est la syntaxe qui préserve le plus la forme des mots ;
- la plupart des projets DGFIP utilise Java (ou tout du moins un langage objet), et cette syntaxe a aussi été retenue dans la charte Java/J2EE. Ainsi, conserver la même convention pour XML assure une meilleure lisibilité de l'association entre un objet métier et sa représentation XML (un champ « monChamp » en Java sera donc associé à un élément « monChamp » dans le document XML représentant cette donnée).

Pour les autres langages (XSLT, XSD, ...) , il faut respecter les conventions établies dans la spécification du langage concerné.

Exemple	Contre exemple
<pre><!-- Nom Xml respectant la syntaxe camelCase --> <nomCompletDuContribuable/> <!-- Nom XSD respectant la syntaxe appropriée --> <xsd:complexType name="PointType"> <!-- Nom XSL, respectant la syntaxe appropriée --> <xsl:template name="common-graph"></pre>	<pre><nom-complet-du-contribuable/> <NOM_COMPLET_DU_CONTRIBUABLE/> <nomcompletducontribuable/></pre>

3.6.5 [NOM-PrefixInterdit] Les noms des éléments et des attributs **ne doivent pas** être préfixés du caractère '_'.

Bien que la spécification XML autorise de préfixer un nom par le caractère '_', la présente charte ne le permet pas, de manière à assurer une meilleure lisibilité du document XML.

Exemple	Contre exemple
<pre><contribuable numeroFiscale="..." /></pre>	<pre><_contribuable _numeroFiscale="..." /></pre>

Remarque : la charte Java/J2EE proscriit explicitement de préfixer un champ de classe par « _ ». Si le document XML représente une sérialisation d'une telle classe, le produit de sérialisation XML doit quand même respecter la règle et ne pas préfixer l'élément ou l'attribut par « _ ».

3.6.6 [NOM-UtilisationSymbole] Les noms des éléments et des attributs **ne doivent pas** contenir de '_', ni de chiffre sauf si ces derniers contiennent une donnée métier.

Pour des raisons de lisibilité, les noms d'éléments ou d'attributs ne doivent pas contenir de « _ ». En outre, l'utilisation de « _ » enfreint les règles [NOM-CamelCase] et [NOM-PrefixInterdit].

Exemple	Contre exemple
<pre><nomCompletDuContribuable/></pre>	<pre><NOM_COMPLET_DU_CONTRIBUABLE/></pre>

De manière générale, un chiffre n'a pas sa place dans un nom d'élément ou d'attribut. L'objectif étant d'obtenir un nom le plus explicite possible, l'utilisation est donc prohibée.

En outre, Il est courant d'utiliser certains chiffres ou symboles pour abrégé un mot :

- « 2 » ou « 4 » pour « de » ou « for » (en anglais) ;
- « @ » pour « chez » ;
- ...

Ce genre de pratique nuit grandement à la lisibilité d'un fichier de données, le rendant non seulement plus cryptique que nécessaire, mais rendant également moins immédiate sa transformation en un modèle objet. Un document XML étant avant tout un format de données, ces abréviations sont donc interdites.

Exemple	Contre exemple
<pre><nomDeLEntreprise> <siègeEnFrance/> </nomDeLEntreprise></pre>	<pre><nom2Entreprise> <siège@France/> </nom2Entreprise></pre>







Cette règle ne proscriit que l'utilisation de chiffres ou de symboles à des fins d'abréviation. Elle n'interdit pas l'utilisation des chiffres dans le cadre de la modélisation de données. Si un chiffre correspond à une donnée « métier », il peut tout à fait être utilisé :

Exemple	Contre exemple
<pre><declarationFiscal id="3"> <formulaire2042C> ... </formulaire2042C> </declarationFiscal></pre>	<pre><declarationFiscal id="3"> <formulaireDeuxZeroQuatreDeuxC> ... </formulaireDeuxZeroQuatreDeuxC> </declarationFiscal></pre>

3.7 Documentation

3.7.1 Finalités, critères de sélection, sources et contraintes

La finalité des règles relatives aux commentaires de traitement est de faciliter la lisibilité et la maintenance par une documentation uniforme et adéquate du code.

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[FORM-COM- Encoding]	Les commentaires doivent respecter l'encodage du document (UTF-8).	<ul style="list-style-type: none"> • DMOD • ADAE - Guide UML-XML ^[1c] 	STD-DGFIP		
[FORM-COM- EntiteAutorises]	Les commentaires ne doivent pas utiliser les entités.	<ul style="list-style-type: none"> • DMOD 	STD-DGFIP		
[FORM-COM- PiedDeDocume ntInterdit]	Il ne doit pas y avoir de commentaire après la fermeture de l'élément racine du document.	<ul style="list-style-type: none"> • DMOD 	STD-DGFIP		

3.7.1.1 [FORM-COM-Encoding] Les commentaires **doivent** respecter l'encodage du document (UTF-8).

De la même manière que le contenu du document XML ne doit pas contenir de caractère interdit par l'encodage UTF-8, les commentaires doivent aussi respecter ce point. La présence de caractères interdits au sein des commentaires peut en effet induire un comportement variable selon le *framework* ou le *parser*.

De plus, si le fichier est chargé dans un éditeur, ce dernier aura vraisemblablement des difficultés à afficher correctement les commentaires.

3.7.1.2 [FORM-COM-EntiteAutorises] Les commentaires ne **doivent pas** utiliser les entités.

Les commentaires peuvent librement contenir des caractères prohibés dans le document en lui même (« <, >, ', ... »). Seule l'utilisation du marqueur de début de commentaire (« <!-- ») est prohibé par la plupart des outils.

Ainsi, les commentaires ne doivent pas contenir d'entité pour représenter les symboles interdits ou les caractères accentués.

Exemple	Contre exemple
<code><!-- Ceci est un commentaire correcte, utilisant des caractères accentués et des symboles interdits tel que <, ' et >. --></code>	<code><!-- Ceci est un commentaire correcte, utilisant des caract&egrave;res accentués et des symboles interdits tel que &lt;; &quote; et &gt;!. --></code>

3.7.1.3 [FORM-COM-PiedDeDocumentInterdit] Il **ne doit pas** y avoir de commentaire après la fermeture de l'élément racine du document.

La spécification XML n'interdit aucunement la présence de commentaire ou de ligne blanche à la suite de la fermeture de l'élément racine. Néanmoins, cette pratique est rare et contre productive. En effet, rechercher des informations supplémentaires sur le document à la fin du fichier n'est pas un réflexe commun et il n'apparaît pas évident de placer des méta données à la fin d'un document.










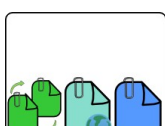

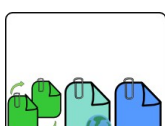

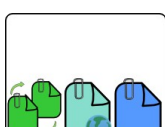
Pour toutes ces raisons, l'utilisation de commentaires en « pied de page » est prohibée.

Exemple	Contre exemple
<code><?xml version="1.0" encoding="UTF-8"> <!-- On peut placer des informations générale sur le document ici --> <racine> ... <!-- on peut aussi ajouter des commentaires au sein du document --> </racine></code>	<code><?xml version="1.0" encoding="UTF-8"> <racine> ... </racine> <!-- Ceci est un commentaire en pied de page, qui ne sera probablement jamais lu dès que la taille du document dépassera 20 lignes... --></code>

3.8 Mise en forme

3.8.1 Finalités, critères de sélection, sources et contraintes

L'ensemble des règles décrites dans ce chapitre porte sur la lisibilité des fichiers XML. Ces règles ne sont applicables donc qu'au document XML visant à être maintenu et/ou manuellement (fichier de configuration, de déploiement, de construction...).

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[FORM-NombreDeColonneDuDocument]	Le document XML doit avoir des lignes d'au plus 120 caractères.	• DMOD	STD-DGFIP		
[FORM-CDATA-PositionSection]	Il est recommandé que les sections CDATA ne soient pas précédées ou suivies d'espace vide.	• DMOD	STD-DGFIP		
[FORM-PRETTYPRINT-PresentationHiérarchie]	Les documents XML doivent être indentés.	• DMOD	STD-DGFIP		
[FORM-PRETTYPRINT-Indentation]	Un document XML, pour être lisible et aisément maintenable, doit être présenté de manière à souligner sa structure hiérarchique.	• DMOD	STD-DGFIP		
[FORM-PRETTYPRINT-RespecterBalisage]	Le pretty-printing d'un élément ne doit pas insérer d'espace entre le nom de l'élément et ses balises de délimitation.	• DMOD	STD-DGFIP		
[FORM-PRETTYPRINT-PositionPremierAttribut]	Le premier attribut de l'élément doit se trouver sur la même ligne que la balise.STD-DGIFIP	• DMOD	STD-DGFIP		
[FORM-PRETTYPRINT-CompatibilitéNonXml]	Il est recommandé de conserver une présentation qui permette l'analyse, ligne à ligne, du fichier.	• DMOD	STD-DGFIP		

3.8.2 Règles générales sur la mise en forme du document

3.8.2.1 [FORM-NombreDeColonneDuDocument] Le document XML **doit** avoir des lignes d'au plus 120 caractères.

La longueur des lignes doit en effet être raisonnable, et permettre une lecture aisée des fichiers dans l'environnement de développement standard, sur un écran de résolution standard.

Il est pour cela préférable d'éviter les lignes trop longues (plus de 120 caractères) qui rendraient difficile la relecture du code.

3.8.3 Règles sur les sections CDATA et PCDATA

Dans un document XML, on distingue deux types de contenus :

1. le contenu destiné à être analysé (« *parsed data* », PCDATA) ;
2. le contenu qui ne doit pas être analysé (« *Character Data* », CDATA).

La présence de contenu PCDATA est implicite, seul le contenu CDATA est indiqué.

Une section CDATA ne doit pas servir à embarquer du texte narratif ! Voir la règle [CONCEP-UtiliseXHTMLPourLeContenuNarratif] sur l'usage de XHTML à ce sujet.

3.8.3.1 [FORM-CDATA-PositionSection] Il est recommandé que les sections CDATA ne soient pas précédées ou suivies d'espace vide.

En termes de présentation, il paraît souvent naturel d'indenter les sections CDATA, ou simplement de les séparer par des espaces ou des sauts de lignes des éléments qui les contiennent. Cette présentation entraîne néanmoins quelques problèmes :

- le parser va systématiquement créer trois sous éléments pour chaque élément contenant la section CDATA :
 - un sous élément pour la section CDATA ;
 - et deux sous éléments de type texte pour contenir les sauts de lignes ou espaces blancs qui précèdent et suivent la section CDATA ;
 cette pratique alourdit légèrement en termes de performance le processus de désérialisation, sans pour autant améliorer grandement la lisibilité ;
- en outre, sous Windows, certains problèmes ont été rencontrés de manière aléatoire lors de l'analyse de documents XML contenant ce genre d'espace ou de saut de lignes.

Pour ces raisons, précéder ou faire suivre les sections CDATA d'espaces ou de sauts de lignes à des seules fins de présentation n'est pas recommandé.

Exemple	Contre exemple
<pre><element><![CDATA[Texte de la section CDATA]]></element></pre>	<pre><element> <![CDATA[Texte de la section CDATA]]> </element></pre>

3.8.4 Règles sur le « *pretty-printing* »

Il est courant de vouloir présenter son XML de manière la plus agréable possible pour une lecture « humaine » du document. Cette pratique se nomme le « *pretty-printing* », et le présent chapitre couvre quelques règles et bonnes pratiques à ce sujet.

3.8.4.1 [FORM-PRETTYPRINT-PresentationHierarchie] Les documents XML doivent être indentés.

Pour faciliter la lecture et la maintenance d'un document XML, il est important de bien indenter ce dernier, pour exposer clairement sa structure hiérarchique.

Exemple	Contre exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <flux ...> <label> <codeEmetteur>P</codeEmetteur> <codeTypeUsager>N</codeTypeUsager> <referenceExterne>REXTU44</referenceExterne> ... </label> <usager> <crpcen>012345</crpcen> <nom>SCP Denis/Denis</nom> <typeUsager>SCP</typeUsager> <perceptionFrais>true</perceptionFrais> <adresseUsager residenceBatiment="..." numeroLibelle="..." lieuDit="..." codePostal="..." .../> <ribUsager> <codeBanque>01234</codeBanque> <codeGuichet>01234</codeGuichet> <compte>CPT0123456789</compte> <domiciliation>Paris CIC</domiciliation> </ribUsager> </usager> ... </flux></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <flux ...> <label><codeEmetteur>P</codeEmetteur><codeT typeUsager>N</codeTypeUsager> <referenceExterne>REXTU44</referenceExterne> ... </label><usager> <crpcen>012345</crpcen><nom>SCP Denis/Denis</nom> <typeUsager>SCP</typeUsager> <perceptionFrais>true</perceptionFrais> <adresseUsager residenceBatiment="..." numeroLibelle="..." lieuDit="..." codePostal="..." .../> <ribUsager><codeBanque>01234</codeBanque> <codeGuichet>01234</codeGuichet> <compte>CPT0123456789</compte> <domiciliation>Paris CIC</domiciliation></ribUsager> </usager> ... </flux></pre>

3.8.4.2 **[FORM-PRETTYPRINT-Indentation]** Un document XML, pour être lisible et aisément maintenable, doit être présenté de manière à souligner sa structure hiérarchique.

Il arrive parfois qu'une ligne d'un document XML soit très longue (cf. [FORM-NombreDeColonneDuDocument]). Alors, il faut effectuer une césure, selon que la ligne contienne un attribut ou un élément.

Dans le cas d'un élément, il est alors recommandé de couper cette ligne en une ou plusieurs en respectant quelques règles :

- couper la ligne après le '>' fermant une nouvelle balise ;
- sauter une ligne ;
- réduire l'indentation d'au moins 2 ou 3 unités (plus si nécessaires) ;
- avant de fermer l'élément, sauter une nouvelle ligne.

Pour un attribut, il est recommandé d'aligner les attributs les uns en dessous des autres, à partir du premier. Si le premier attribut dépasse déjà la limite, on utilise alors la stratégie décrite pour les éléments, sur son élément porteur.

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<flux xmlns="http://flux.teleactes.copernic.finances.gouv.fr/v20081">
  ...
  <Acte-vente xmlns="http://teleactes.real.not/v20081">
    ...
    <formalite>
      ...
      <immeuble-acte>
        <immeuble identifiant="IMM0000001" nature-immeuble="parcelle sans EDD">
          <volume numero="1">
            <!-- les attributs de cadastre ont été alignés pour effectuer
                  la césure de la ligne -->
            <cadastre code-departement="060"
                      code-commune="012"
                      commune="Beausoleil"
                      section="AA"
                      numero="0010"/>
          </volume>
          <lot numero="1L" surface="125.00"/>
          ...
          <autre-formalite-publiee code-ch-depot="0604P04">
            <reference>
              <!-- le contenu de l'élément reference-formalite-type3 a
                    été reculé pour éviter la césure de son contenu -->
              <reference-formalite-type3>
                Référence effet relatif avant 1956 et usucapion ...
              </reference-formalite-type3>
            </reference>
          </autre-formalite-publiee>
        ...
      ...
    ...
  ...

```

Remarque : L'unité d'indentation doit être constituée de 4 espaces. Il n'est pas autorisé d'utiliser les tabulations pour l'indentation. On remarquera, que pour des raisons de présentation, les exemples de ce document ne peuvent pas toujours suivre cette règle.

3.8.4.3 **[FORM-PRETTYPRINT-RespecterBalisage]** Le pretty-printing d'un élément **ne doit pas** insérer d'espace entre le nom de l'élément et ses balises de délimitation.

Pour des raisons d'optimisation, certains développeurs préfèrent formater leurs données XML de la manière montrée par le contre exemple ci dessous.

Exemple	Contre exemple
<pre><element> <sousElement>contenu</sousElement> </element></pre>	<pre><element ><sousElement>contenu</sousElement> </element></pre>

La syntaxe du contre exemple ci-dessus est valide au sens XML, sans créer d'élément texte supplémentaire. En effet, un *framework* peut représenter le contenu de l'élément <element> ainsi :

- élément de type texte contenant : « \n\t » (un retour chariot et une tabulation) ;
- un élément associé à « <sous-element> » et son contenu ;
- un élément de type texte contenant : « \n » (un retour chariot).

L'*overhead* rajouté par ces zones de texte inutiles ne nuit pas réellement aux performances.

Néanmoins, dans le contexte du SI de la DGFiP, cette option n'a pas été retenue pour les raisons suivantes :

1. elle nuit grandement à la lisibilité, un des points essentiels de XML ;
2. cette syntaxe peut facilement entraîner des erreurs typographiques qui transformeraient un document bien formé en un document invalide.

3.8.5 **[FORM-PRETTYPRINT-PositionPremierAttribut]** Le premier attribut de l'élément **doit** se trouver sur la même ligne que la balise.

Pour assurer une bonne lisibilité, il est important de bien structurer la présentation des attributs d'un élément XML.

Exemple	Contre exemple
<pre><element premier_attribut="valeur"> <sous-element> contenu </sous-element> </element></pre>	<pre><element premier_attribut="valeur"> <sous-element> contenu </sous-element> </element></pre>

3.8.6 **[FORM-PRETTYPRINT-CompatibilitéNonXml]** Il est recommandé de conserver une présentation qui permette l'analyse, ligne à ligne, du fichier.

Cette pratique facilite le travail des administrateurs système qui doivent manipuler des documents XML, et qui sont habitués aux outils unix tel que « *sed, grep* » ou « *awk* ». En outre, cette pratique améliore la lisibilité et l'analyse des fichiers XML.













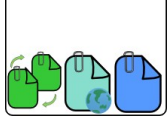

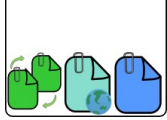

De plus, l'alternative XML à ce genre de solution, comme « *xsltproc* », n'est pas simple à mettre en place. En effet, il n'est pas toujours aisé d'écrire une requête XPath pour retrouver une information précise dans un document XML.



Exemple	Contre exemple
<pre><element attribut="valeur"> <sous-element> contenu </sous-element> </element></pre>	<pre><element attribut="valeur"><sous-element> contenu </sous-element></element></pre>

3.9 Conception de document XML

3.9.1 Finalités, critères de sélection, sources et contraintes

Ce chapitre regroupe un ensemble de règles sur la structuration d'un fichier XML.

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[CONCEP- MaxAttributs]	Un élément doit avoir moins de 6 attributs.	• DMOD	STD-DGFIP		
[CONCEP- DonneeUnique]	Un attribut doit représenter une donnée unique et atomique, sinon la donnée doit être représentée par un élément.	• Effective XML ^[4] • DMOD	STD-DGFIP		
[CONCEP- NomElementPe re]	Un attribut ne doit pas avoir le même nom que son élément.	• DMOD	STD-DGFIP		
[CONCEP- ProfondeurArb orescence]	Une arborescence XML ne doit pas être plus profonde que 10 niveaux.	• DMOD	STD-DGFIP		
[CONCEP- ElementListe]	Les éléments « liste » doivent être utilisés pour encadrer un ensemble d'éléments identiques successifs.	• Effective XML ^[4] • DMOD	STD-DGFIP		
[CONCEP- ConteneurListe]	Le nom des éléments «liste», contenant une série d'éléments identiques, doit toujours être au pluriel.	• DMOD	STD-DGFIP		
[CONCEP- ValeurAtomiqu e]	Les données d'un champ XML doivent être aussi atomiques que l'exige la modélisation.	• Effective XML ^[4] • DMOD	STD-DGFIP		
[CONCEP- PasDeBinaire]	Les données binaires d'un document XML doivent être externalisées.	• Effective XML ^[4]	STD-DGFIP		
[CONCEP- UtiliseXHTML PourLeContenu Narratif]	Il est recommandé d'utiliser XHTML pour structurer les données de type texte dans un document.	• Effective XML ^[4]	STD-DGFIP		
[CONCEP- QuandUtiliserC DATA]	Il est recommandé de n'utiliser les sections CDATA dans un document XML que si nécessaire	• Effective XML ^[4]	STD-DGFIP		

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[CONCEP-DocumentIndépendant]	[CONCEP-DocumentIndépendant] Un document XML doit être indépendant en ayant aucune référence externe aux documents autres que les XSD et des données binaires.	<ul style="list-style-type: none"> Effective XML^[4] 	STD-DGFIP		

3.9.2 Métriques de conception

3.9.2.1 [CONCEP-MaxAttributs] Un élément doit avoir moins de 6 attributs.

Les attributs permettent d'ajouter des informations, le plus souvent des méta informations, à un élément. Il est important de limiter le nombre de ces attributs, qui ne sont pas un mécanisme de modélisation très souple.

Exemple	Contre exemple
<pre><etat-civil code-commune-naissance-topad="012"> <nom>Mathieu</nom> <date-naissance>1942-12-31</date> <commune-naissance>Paris</commune-naissance> <departement-naissance>75</departement-naissance> <libelle-pays-naissance>France<libelle-pays-naissance> </etat-civil></pre>	<pre><etat-civil civilite="monsieur" nom="Mathieu" date-naissance="1942-12-31" commune-naissance="Paris" code-commune-naissance-topad="012" departement-naissance="75" libelle-pays-naissance="France"/></pre>

3.9.3 Relation attribut-élément

3.9.3.1 [CONCEP-DonneeUnique] Un attribut doit représenter une donnée unique et atomique, sinon la donnée doit être représentée par un élément.

Une donnée peut être modélisée dans un document XML sous forme d'élément ou d'attribut.

Il est souvent tentant de modéliser la donnée par un attribut pour des raisons de simplicité. Néanmoins, il est important de prendre le temps d'étudier les deux points qui suivent, lorsque l'on décide de représenter une donnée par un attribut XML.

3.9.3.1.1 Est-ce que la donnée est unique ?

Un attribut XML est obligatoirement unique. Aucun élément ne peut avoir deux attributs qui portent le même nom. Il est donc nécessaire, avant de représenter une donnée sous forme d'attribut, que cette dernière soit toujours unique, et qu'à aucun moment on ne se trouve à associer un élément XML à deux instances différentes de cette information.

Dans l'exemple ci-dessous, la modélisation proposée en contre exemple ne permettra pas d'avoir des éléments « personne » ayant plusieurs prénoms. Il est probable qu'au démarrage ce point ne pose pas particulièrement de problèmes, mais sur le long terme, par exemple quand le système entrera en interaction avec d'autres, cette limitation deviendra probablement une source de problèmes.

Exemple	Contre exemple
<pre><personne> <prenoms> <prenom>Romain</prenom> </prenoms> ... </personne></pre>	<pre><personne prenom="Romain"> ... </personne></pre>

3.9.3.1.2 Est-ce que la donnée est atomique ?

Ceci fait écho à la règle [CONCEP-ValeurAtomique]. Si la donnée représentée sous forme d'attribut est trop structurée, elle nécessitera des algorithmes de traitements supplémentaires, qui sont inutiles et rendent plus complexes le fonctionnement de l'application.

Si la donnée modélisée est structurée, il est donc plus simple et plus pertinent de la modéliser sous forme de sous élément plutôt que d'attribut.

Exemple	Contre exemple
<pre><personne> <adresse> <numero>4</numero> <rue nature="avenue">montaigne</rue> </adresse> ... </personne></pre>	<pre><personne adresse="4, avenue Montaigne"> ... </personne></pre>

3.9.3.1.3 [CONCEP-NomElementPere] Un attribut **ne doit pas** avoir le même nom que son élément.

Cette règle fait écho à la règle Java/J2EE [NOM-15]. Un élément dont un attribut porte le même nom nuit à la lisibilité du document XML.

Exemple	Contre exemple
<pre><type nature="..." > ... </type></pre>	<pre><type type="..."> ... </type></pre>

3.9.4 Hiérarchie et arborescence

3.9.4.1 [CONCEP-ProfondeurArborescence] Une arborescence XML **ne doit pas** être plus profonde que 10 niveaux.

La modélisation d'un ensemble de données sous forme d'arbre XML ayant une profondeur de plus de 10 niveaux est douteuse. En outre, un document XML d'une telle profondeur crée aussi des risques en termes de maintenance et de performance.

Cette règle ne s'applique qu'aux documents modélisés pour les projets DGFIP, elle ne s'applique pas aux langages XML normalisés (tels que SVG ou SOAP par exemple) ou encore au format XML d'applications ou outils tiers (Maven, Jboss,...).

Contre exemple

```

<flux ...>
  ...
  <acteVentes>
    <acteVente>
      <formalite>
        <immeuble-acte>
          <immeuble ...>
            <autreFormalitePubliee ...>
              <reference>
                <referenceFormaliteType>
                  <annee>2001</annee>
                  <lettreEnlissement>P</lettreEnlissement>
                  <numeroEnlissement>100</numeroEnlissement>
                </referenceFormaliteType>
              </reference>
            </autreFormalitePubliee>
          </immeuble>
        </immeuble-acte>
      </formalite>
    </acteVente>
  </acteVentes>
</flux>

```

Cette règle ne s'applique que sur les éléments modélisés par l'applicatif. Si le document contient des données XML modélisées par un autre applicatif ou une application tierce, il est probable que sa structure atteindra plus de 10 niveaux de profondeur. Dans ce contexte, il faut utiliser un préfixe adapté à l'espace de noms inclus.

Exemple

```

<texte-inscription>
  <dsig:Signature Id="sigInscription">
    <dsig:SignedInfo...
  </dsig:Signature>
</texte-inscription>

```

3.9.4.2 **[CONCEP-ElementListe]** Les éléments « liste » **doivent** être utilisés pour encadrer un ensemble d'éléments identiques successifs.

Si des éléments « dossier » sont régulièrement regroupés en ensembles de dossiers, l'élément « <dossiers> » a un sens. A l'inverse, si une suite de dossiers ne dépend pas des circonstances, l'élément « <dossiers> » ne représente aucune réalité au niveau de la modélisation des données.

Exemple	Contre-Exemple
<pre> <dossiers> <dossier/> <dossier/> </dossiers> </pre>	<pre> <dossier/> <dossier/> </pre>

Cette règle correspond au *pattern* XML « Collection Element ».

3.9.5 **[CONCEP-ConteneurListe]** Le nom des éléments « liste », contenant une série d'éléments identiques, **doit** toujours être au pluriel.

Cette pratique améliore la lisibilité du document XML et facilite aussi son traitement par les programmes.

Exemple	Contre exemple
<pre><dossiers> <dossier/> <dossier/> </dossiers></pre>	<pre><armoire> <dossier/> <dossier/> </armoire></pre>

3.9.6 Données modélisées

3.9.6.1 [CONCEP-ValeurAtomique] Les données d'un champ XML **doivent** être aussi atomiques que l'exige la modélisation.

Un document XML, par essence, contient un ensemble de données *structurées*. Les différents *frameworks* facilitent grandement l'accès à n'importe quelle donnée, tant qu'elle est incluse dans un élément ou un attribut. Néanmoins, dans certains cas, la tentation est forte chez un concepteur de ne pas tout modéliser par une structure XML.

Un exemple concret est la modélisation du nom d'un individu. Dans la plupart des cas une balise « <nom> » contenant le nom de l'individu semble suffisant : « <nom>Romain PELISSE</nom> ».

Si le programme client doit séparer le nom et le prénom, il suffit d'utiliser un « *tokenizer* » pour séparer le nom du prénom. Mais cette approche a très vite des limites :

Contre exemple
<pre><nom>Romain Sébastien PELISSE</nom> <nom>Maurizio De Cecco</nom> <nom>Docteur Malik Salheb</nom></pre>

Ce contre exemple souligne une faiblesse dans la modélisation XML de la donnée. La donnée n'a pas été modélisée de manière assez fine. Un nom n'est pas une donnée atomique. Il contient une structure sous-jacente.

Exemple	Contre exemple
<pre><nom> <prenom>Malik</prenom> <nomDeFamille>Salheb</nomDeFamille> <titre>Docteur</titre> </nom> ... <nom> <prenom>Maurizio</prenom> <nomDeFamille>De Cecco</nomDeFamille> <titre/> </nom> ... <nom> <prenom>Romain Sébastien</prenom> <nomDeFamille>Pelisse</nomDeFamille> <titre/> </nom></pre>	<pre><nom>Romain PELISSE</nom> <nom>Docteur Malik Salheb</nom> <nom>Maurizio De Cecco</nom></pre>

Attention, il ne faut pas non plus aller trop loin. Dans l'exemple ci-dessus, nous avons choisi de garder une sémantique simple, dans le sens où l'élément « `<prenoms>` » contient l'ensemble des prénoms d'un individu. En effet, il ne semble pas très pertinent, ni pragmatique d'alourdir notre modélisation ainsi :

Exemple	Contre-Exemple
<code><prenoms>Romain Sébastien</prenoms></code>	<pre> <prenoms> <prenom>Romain</prenom> <prenom>Sébastien</prenom> </prenoms> </pre>

De même, si la donnée est structurée dans un format bien reconnu (comme une date au format jour/mois/années), il n'est pas très pertinent de créer une structure aussi complexe que ci-dessous, qui est de surcroît plus difficile à interpréter et demande plus de travail au développeur :

Exemple	Contre-Exemple
<code><date>2008/08/02</day></code>	<pre> <date> <annee>2008</annee> <mois>08</mois> <jour>02</jour> </date> </pre>

La limite de l'atomicité d'une donnée est avant tout définie par les besoins de l'applicatif. L'objectif est d'éviter que le module applicatif effectue un traitement supplémentaire pour restructurer une donnée faiblement structurée au sein du document XML. Il ne s'agit pas d'obtenir la modélisation la plus fine de manière absolue. D'une manière générale, l'atomicité recherchée doit rester au même niveau que la vue modélisation de l'application.

Pour reprendre l'exemple ci dessus, si l'applicatif n'utilise le nom d'un individu que comme un tout unique, la modélisation de ce dernier par une simple balise « `<nom>` » est amplement suffisante. A l'inverse, si le module applicatif doit manipuler ses différents prénoms et titres, il est impératif de les modéliser de manière détaillée et de laisser la charge de sérialisation/désérialisation de cette structure au *frameworks* XML.

3.9.6.2 [CONCEP-PasDeBinaire] Les données binaires d'un document XML **doivent** être externalisées.

Un document XML est un document texte. Insérer des données binaires au sein d'un élément est donc en contradiction avec la nature et les objectifs de XML. Il est donc préférable de ne pas inclure ce type de données directement dans le document.

De plus, si le flux binaire n'est pas proprement encadré par une section CDATA, le framework peut éventuellement considérer le fichier comme invalide s'il contient des caractères interdit par XML (« `<`, `>`, ... »).

Cette règle s'applique aussi au contenu encodé en base 64, tel que les signatures.

Exemple	Contre exemple
<pre> <dsig:KeyInfo Id="sigInscription-KeyInfo"> <dsig:KeyValue> <dsig:RSAKeyValue> <dsig:Modulus src="modulus.dat"/> </dsig:RSAKeyValue> </dsig:KeyValue> <dsig:X509Data> <dsig:X509Certificate src="certificat.cem"/> </dsig:X509Data> </dsig:KeyInfo> </pre>	<pre> <dsig:KeyInfo Id="sigInscription-KeyInfo"> <dsig:KeyValue> <dsig:RSAKeyValue> <dsig:Modulus>wVtCyicks29uNdDvAx1zIPsZ rAT4VuaLAZjHfTI39cwV5jFuAOFPlDAzhmkTy1qw6KA47D5G 4WLk 5dykk0g2ntpzEalKhH800nXaXLT3B6+oaekRbEKNU68t6NX ur6OkskMO9Fa/IT11JCS14UzUpPm Z06WaGFUBoEXbSdR5SE=</dsig:Modulus> <dsig:Exponent>AQAB</dsig:Exponent> </dsig:RSAKeyValue> </dsig:KeyValue> <dsig:X509Data> <dsig:X509Certificate>MIIDtzCCAqGgAwIBAgIC LnwwCwYJKoZIhvcNAQEFMG0xCzAJBgNVBAYTAmZyMSMwIQYD VQQKDBpQ cm9mZXNzaW9ucyBSw6lnbGVtZW50w6llczERMA8GA1UECxMI Tm90YWlyZXNxFzAVBgNVBAsMDkFDIGTDqWzD IGTDqWzDqWd1w6llMQ0wCwYDVQQLEwRSRUFB4XDTA1MTEy MjA5MzkwNVoXDTA3MTEyMjA5Mzkw NVowgaYxCzAJBgNVBAYTAmZyMSMwIQYDVQQKDBpQcm9mZXNz aW9ucyBSw6lnbGVtZW50w6llczER MA8GA1UECxMITm90YWlyZXNxFzAVBgNVBAsMDkFDIGTDqWzD qWd1w6llMQ0wCwYDVQQLEwRSRUFB MTcwGAYDVQQDExFtViAyICgzOTkwMDQwMTA3KTA3BgkqhkiG 9w0BCQEEMDn2MkBpbmZvbGllLmZy MIGdMASGCSqGSIb3DQEBAQOBjQAwgYkCgYEAwVtCyicks29u NdDvAx1zIPsZrAT4VuaLAZjHfTI3 9cwV5jFuAOFPlDAzhmkTy1qw6KA47D5G4WLk5dykk0g2ntpz EalKhH800nXaXLT3B6+oaekRbEKNU QU68t6NXur6OkskMO9Fa/IT11JCS14UzUpPmZ06WaGFUBoEX bSdR5SECAwEAaOBsDCBrTAdBgNV HQ4EFgQUIwL/9UDvAeq/lsm2EAQSWFP0mUwMwYDVR0RBCww KoEOc3YyQGluZm9saWIuZnKgGAYI KoF6AUwCBQGgDBQKMzk5MDA0MDEwMzA0BgNVHQ8BAF8EBAMC BkAwHwYDVR0jBBGwFoAUToyh6C66 3tCmqS4hqPXWVwSB9ZMwGwYDVR0gBBQwEjAQBg4qgXoBTgEB AwEDAgUCAjA3BgNVHRMEAjaAMAsG CSqGSIb3DQEBAQQAQEAghxn1Z1UR4RNUUVrghXl1dsQgkUA 9vS6tn01Nq51oSzOCJBgv9/yhLXr mZ0GqzIoCqN1151fZ4C6tuYp7Xuhv6lc9wNt9uT/LZNPlyxd 50CRggRN7K7Xwc0y/eyfOe88dQgx k9g2kvwdMP5QhoLCiF4YEAYmBakTYWL8mzRG1UIY7MlrPnwN gV188F8EMQ8D4Xzo62aXvKdxmAYa yLTOM+vSHxwB7aJUpuQUAg1z0RJM9dpreiJFldHXUq+1tU WeRCHxfEdRto+9uS8RNeVSAlBpvh 9rsZsKgg13/RvyoilIikNJEytcFAjellU+cFkVVDLg+pVScW /xQVg6xUrg==</dsig:X509Certificate> </dsig:X509Data> </dsig:KeyInfo> </pre>

3.9.6.3 [CONCEP-UtiliseXHTMLPourLeContenuNarratif] Il est recommandé d'utiliser XHTML pour structurer les données de type texte dans un document.

Un élément XML peut encapsuler un contenu narratif. Ce type d'information est simplement un texte destiné à être lu par un être humain, comme, par exemple, un extrait de texte juridique. Néanmoins, il ne peut se modéliser comme une simple zone de texte, car il dispose d'une structure sous-jacente trop complexe.

Ce genre de donnée embarquée par une application XML doit être modélisé de manière structurée (voir les règles [CONCEP-ValeurAtomique] et [CONCEP-DocumentIndépendant]).

Dans ce contexte, il est rarement pertinent que le projet définisse sa propre sémantique XML pour structurer le contenu narratif. Ainsi, il est recommandé d'utiliser la syntaxe XHTML qui est non seulement valide, mais aussi normalisée par le W3C et surtout dédiée à la structuration de document narratif.

Exemple

```
<partieLegale>
  <licence>
    <html xmlns="http://www.w3.org/1999/xhtml">
      <body>
<h1>Licence Publique Générale GNU</h1>
<h2>Préambule</h2>

<p>La Licence Publique Générale GNU (<em lang="en">"GNU General Public
License"</em>) est une licence libre, en <em lang="en">"copyleft"</em>,
destinée aux œuvres logicielles et d'autres types de travaux.</p>
...
<p>Quand nous parlons de logiciel libre (<em lang="en">"free"</em>), nous
nous référons à la liberté (<em lang="en">"freedom"</em>), pas au prix. Nos
Licences Publiques Générales sont conçues pour assurer que vous ayez la
liberté de distribuer des copies de logiciel libre (et le facturer si vous le
souhaitez), que vous receviez le code source ou pouviez l'obtenir si vous le
voulez, que vous pouviez modifier le logiciel ou en utiliser toute partie dans
de nouveaux logiciels libres, et que vous sachiez que vous avez le droit de
faire tout ceci.</p>
...
<h2><a name="terms"></a><big>ERMES ET CONDITIONS</h3>

<h3><a name="section0"></a>Article 0. Définitions.</h4>
...
      </body>
    </html>
  </licence>
</partieLegale>
```

Contre Exemple

```
<partieLegale>
  <licence><![CDATA[
Licence Publique Générale GNU

  Préambule

La Licence Publique Générale GNU ("GNU General Public
License") est une licence libre, en "copyleft",
destinée aux œuvres logicielles et d'autres types de travaux.
...
Quand nous parlons de logiciel libre ("free"), nous
nous référons à la liberté ("freedom"), pas au prix. Nos
Licences Publiques Générales sont conçues pour assurer que vous ayez la
liberté de distribuer des copies de logiciel libre (et le facturer si vous le souhaitez), que
vous receviez le code source ou pouviez l'obtenir si vous le voulez, que vous pouviez
modifier le logiciel ou en utiliser toute partie dans de nouveaux logiciels libres, et que
vous sachiez que vous avez le droit de faire tout ceci.
...

TERMES ET CONDITIONS

  Article 0. Définitions
  ...
]]></licence>
</partieLegale>
```


Remarque : la règle [GEN-XHTMLStrict] de la charte HTML/CSS indique que l'utilisation de la variante « strict » du XHTML 1.0 est **obligatoire**. Même si le contenu narratif intégré n'a pas pour objet d'être publié, il est **fortement recommandé** d'utiliser du XHTML Strict.

3.9.6.4 [CONCEP-QuandUtiliserCDATA] Il est recommandé de n'utiliser les sections CDATA dans un document XML que si nécessaire

Cette recommandation est issue d'un constat simple. Les sections CDATA permettent de :

1. insérer du texte contenant des caractères interdits par la syntaxe XML ('<', '>', et '&') ou des caractères accentués ;
2. associer à un élément un ensemble de sous éléments XML sans qu'il ne soit analysé par le *framework*.

Or, la règle [PROLOGUE-encodage], qui préconise l'utilisation de UTF-8, et la règle [XML-ENC-EntitesNecessaires], qui préconise l'utilisation des entités pour représenter les symboles utilisés par la syntaxe XML, aboutissent à rendre caduc le premier cas d'utilisation des CDATA.

Le second cas d'utilisation des CDATA visant à embarquer un fragment XML sans qu'il ne soit analysé est, au mieux, douteux. Dans une logique déjà évoquée dans la règle précédente ([CONCEP-UtiliseXHTMLPourLeContenuNarratif]), il semble plus pertinent d'utiliser les espaces de nommage pour embarquer un tel contenu.

Exemple	Contre exemple
<pre><partieLegale> <licence> <html xmlns="http://www.w3.org/1999/xhtml"> <body> <h1>Licence Publique Générale GNU</h1> ... </partieLegale></pre>	<pre><partieLegale> <licence><![CDATA[<html> <body>Licence Publique <h1>Générale GNU</h1> ...]]> </partieLegale></pre>

Le seul cas d'utilisation logique de section CDATA est celui visant à embarquer au sein du document un contenu doté de balise mais qui n'est pas bien formé. Il s'agit donc d'un ensemble de texte et de balises ouvertes, qui ne sont pas systématiquement fermées comme en HTML.

Exemple	Contre exemple
<pre><partieLegale> <licence><![CDATA[[<body> <p>Licence Publique Générale GNU<p>]]> ... </partieLegale></pre>	<pre><partieLegale> <licence> <!-- document mal formé ! --> <body> <p>Licence Publique Générale GNU<p> </partieLegale></pre>

Ainsi, dans le contexte des projets DGFIP, il n'existe que peu de cas d'utilisations pertinents des sections CDATA.

3.9.7 Intégrité du document

3.9.7.1 [CONCEP-DocumentIndépendant] Un document XML **doit** être indépendant en ayant aucune référence externe aux documents autres que les XSD et des données binaires.

Pour des raisons de simplicité (mais aussi de sécurité), un document XML doit être le plus complet possible, et ne doit nécessiter aucune information externe, à l'exception de son schéma de validation et des données binaires qu'il référence ([CONCEP-PasDeBinaire]). Cette règle vise aussi à assurer l'indépendance d'un document XML.

3.10 Utilisation des *patterns* et *anti-patterns* XML

3.10.1 Finalité, critères de sélection, sources et contraintes

Un certain nombre de *design patterns*, associé aux technologies XML, a été identifié et synthétisé par différents acteurs de l'industrie. Le chapitre en présente une synthèse et indique dans quelle mesure ils sont pertinents dans le contexte de la DGFIP.

Ces *design patterns* XML ne sont pas aussi reconnus et universels que leurs équivalents objets, ils ne sont donc pas aussi familiers au développeur et il est probable que peu de concepteurs de documents XML les connaissent. Néanmoins, la plupart de ces *design patterns* restent des bonnes pratiques qui méritent d'être étudiées et, s'ils sont appropriés, appliqués lors de la conception de documents XML.

A l'inverse, certains de ces *patterns* sont en contradiction avec des règles de cette charte. Il est donc important de les signaler. Ces cas sont clairement identifiés dans ce chapitre.

3.10.2 *Patterns* associés aux règles déjà citées de la charte

Parmi les *patterns* identifiés, un certain nombre correspond à des règles définies dans cette présente charte. Ainsi, le pattern « *Universal Root* » (« Racine Universelle ») recommande l'utilisation du pattern XSD « *VenetianBlind* » tel que défini par la règle [XSD-Pattern-VenetianBlind].

Le « *Collection Element* » (« L'élément de collection ») est un *pattern* qui correspond à la règle [CONCEP-ElementListe]. De même, « *Reuse Document Types* », qui encourage la réutilisation de schémas XML existants, correspond à la règle [CONCEP-UtiliseXHTMLPourLeContenuNarratif].

Le « *Short Understandable Name* » est plus une bonne pratique qui fait écho aux règles de nommage ([NOM-NomExplicite]) de cette charte.

3.10.3 *Patterns* en contradiction avec les règles déjà citées de la charte

A l'inverse, le « *Metadata in Separate Document* » (« Placer les métadonnées dans un document à part »), recommande de séparer les métadonnées, si elles sont conséquentes, dans un document XML à part. Il est donc en contradiction avec la règle [CONCEP-DocumentIndépendant], qui impose d'avoir des documents le plus complets et cohérents possible.

3.10.3.1 « *Patterns* » recommandant l'utilisation des DTDs

L'ensemble des « *pattern* » cités ci dessous préconise des solutions utilisant les mécanismes offerts par les DTDs. Ils sont en contradiction avec la règle [XSD-PasDeDTD] qui préconise de ne pas utiliser les DTDs.

Le « *Optional Container Element* » (« Élément conteneur optionnel ») recommande de réduire la complexité des DTDs du document en utilisant des éléments optionnels. Il est donc en contradiction avec la règle [XSD-PasDeDTD].

Dans le même ordre d'idée, « *Choice Reducing Container* » (« Conteneur à réduction de choix ») est un « *pattern* » qui recommande de réduire la sémantique des DTDs et de faciliter les regroupements. En outre, cette règle recommande, de manière explicite, de placer des informations de définition de contenu dans un schéma de validation, ce qui entre en conflit avec la règle [CONCEP-DocumentIndépendant] et [XSD-TYPE-PasDeValeursParDefautNiFixe].

Le « *Extensible Content Model* » (« Contenu extensible ») recommande d'utiliser les DTDs pour permettre aux utilisateurs du document XML d'étendre sa sémantique. Il est donc en contradiction avec la règle [XSD-PasDeDTD], mais aussi avec le principe général de la règle [XSD-TYPE-PasDeSubstitution] qui n'autorise pas les schémas extensibles. La même remarque s'applique au « *Generic Element* » qui décrit des éléments extensibles. Il est donc en contradiction avec la règle [XSD-PasDeDTD], mais aussi avec le principe général de ne pas autoriser les schémas extensibles.

Enfin, les *patterns* « *Consistent Element Set* » (« Jeu d'élément cohérent »), « *Content Type Label* » et « *Parallel Design* » (« Conception en parallèle ») qui décrivent, respectivement, comment implémenter dans une DTD une série de sous éléments réutilisables et comment structurer une DTD, sont donc aussi en contradiction avec la présente charte.

3.10.3.2 Autres *patterns* en contradiction

Le « *Role Attributes* » recommande d'utiliser un attribut « Role » pour permettre aux auteurs des document XML de catégoriser certains éléments du document. Cette utilisation des attributs pour définir des relations entre entités n'est pas en cohérence avec les préceptes du Guide UML-XML de l'ADAE.

Le « *Multi Root Document Types* » adresse les situations où deux document différents (où l'élément racine est différent) partagent un ensemble de donnée similaires. Le *pattern* recommande de laisser les documents séparés, ce qui est en contradiction avec la règle [CONCEP-DocumentIndépendant].

3.10.3.3 *Patterns* hors périmètre de la charte

Malgré leurs noms, « *XML In Out Tray* » et « *The XML Acceptor Pattern* », ces *patterns* sont plus des *patterns* d'architecture applicative que des *patterns* spécifiques à XML. En conséquence, ils ne sont pas dans le périmètre du présent document.

De même, le « *External Assistant* », qui préconise l'utilisation d'appel à des classes Java au sein de XSL pour améliorer la modularité d'une architecture applicative, et le « *XML Mediator* », qui recommande plusieurs principes d'architecture pour organiser des données XML issues de plusieurs sources, sortent aussi du périmètre du présent document.

Le « *UseXML* » est un *pattern* qui vise à définir les usages les plus pertinents de XML, selon les enjeux. Il sort donc du périmètre de cette charte en tant que tel, mais son étude reste tout à fait valide en phase d'analyse et de conception d'un applicatif. En effet, il permettra de déterminer si le XML est adapté à l'ensemble des données modélisées.

3.10.3.4 *Patterns* discutables

Comme expliqué au début de ce chapitre, les *patterns* XML ne sont pas aussi établis et reconnus que les *patterns* Objet. Il existe donc une série de *patterns* peu pertinents et peu structurants dans le contexte de cette charte, ils n'ont donc pas été retenus :

- « *Multiple Document Types* » ;
- « *Separate metadata and data* » ;
- « *Container Element* » ;
- « *Domain Element* ».

3.10.4 Règles issues des *patterns* XML

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[Pattern-HeadBody]	Pour séparer, au sein d'un document XML, les métadonnées des données, il est recommandé de les placer dans un entête au début du document.	• DMOD	STD	★	
[Pattern-MetaDataFirst]	Pour toute information dotée de métadonnées, il est recommandé d'exposer les métadonnées avant l'information.	• DMOD	STD	★	
[Pattern-Envelope]	Il est recommandé d'encapsuler les parties dynamiques d'un document XML dans une enveloppe.	• DMOD	STD	★	
[Pattern-FlyWeight]	Il est recommandé de centraliser les informations et d'utiliser les attributs « ID » et « IDREF ».	• DMOD	STD	★	
[Pattern-Marketplace]	Il est recommandé d'étiqueter les données pouvant être catégorisées de différentes manières plutôt que d'essayer de les structurer de manière hiérarchique.	• DMOD	STD	★	
[Pattern-ReferencedNote]	Il est recommandé d'isoler les annotations sur un contenu dans un élément séparé et les référencer à l'aide des attributs spéciaux « ID » et « IDREF ».	• DMOD	STD	★	
[Pattern-InformationGrouping]	L'utilisation de « Information Grouping » est recommandée dès lors qu'il s'agit de générer un document XML, à l'aide de XSL, à partir d'un autre.	• DMOD	STD	★	

3.10.4.1 **[Pattern-HeadBody]** Pour séparer, au sein d'un document XML, les métadonnées des données, il est recommandé de les placer dans un entête au début du document.

Si un document XML contient un vaste ensemble de métadonnées, il est pertinent de les structurer. Le « *pattern* » « *Head Body* » (« Tête et corps ») recommande en effet de reprendre l'architecture d'une page HTML, et donc de séparer les métadonnées dans une balise enfant (« head ») de l'élément racine, et de placer le contenu dans une autre balise enfant de l'élément racine (« body »).

Ce « *pattern* » s'applique quand une métadonnée doit être incluse comme un élément (et non simplement comme un attribut).

3.10.4.2 **[Pattern-MetaDataFirst]** Pour toute information dotée de métadonnées, il est recommandé d'exposer les métadonnées avant l'information.

Dès l'instant où le document XML contient des métadonnées propre aux données modélisées, il convient de l'exposer au plus vite au « *framework* », soit en les plaçant dans les premiers attributs ou dans les premiers éléments fils.

Exposer rapidement ces métadonnées permet d'obtenir de meilleure performance avec la plupart des frameworks.

3.10.4.3 **[Pattern-Envelope]** Il est recommandé d'encapsuler les parties dynamiques d'un document XML dans une enveloppe.

Le *pattern* « *Envelope* » recommande d'utiliser un élément « envelope » pour englober les parties dynamique du document. Ainsi, si une partie des données modélisées n'a pas une structure fixe, il est possible d'utiliser différents *namespaces*, placés sur cet élément « envelope » :

Exemple

```
<e:Envelope xmlns:e="envelope1.xsd">
  <!-- Premier type de contenu possible -->
</e:Envelope>
<e:Envelope xmlns:e="envelope2.xsd">
  <!-- Deuxième de contenu variable -->
</e:Envelope>
<e:Envelope>
  <!-- Contenu variable -->
</e:Envelope>
```

Ce *pattern* s'applique dès lors qu'un sous ensemble des données modélisées est trop dynamique ou variable pour être modélisé dans une structure hiérarchique fixe.

3.10.4.4 **[Pattern-FlyWeight]** Il est recommandé de centraliser les informations et d'utiliser les attributs « ID » et « IDREF ».

Le *pattern* « *Flyweight* » recommande de centraliser l'information redondante en un seul endroit et de créer une référence vers cette section là où c'est nécessaire.

Pour mettre en place ces références, le *pattern* propose plusieurs mécanismes . Dans le cadre des projets de la DGFIP, la solution recommandée est l'utilisation des attributs spéciaux « ID » et « IDREF ».

Exemple

```
<document>
  <texte id="titre">Mon document</texte>
  <titre IDREF="titre"/>
  <h1 IDREF="titre"/>
</document>
```

Ce *pattern* s'applique dès lors qu'une information est répétée à plusieurs endroits dans le document XML.

3.10.4.5 [Pattern-Marketplace] Il est recommandé d'étiqueter les données pouvant être catégorisées de différentes manières plutôt que d'essayer de les structurer de manière hiérarchique.

Le *pattern* « *Marketplace* » recommande de ne pas essayer de structurer des données qui peuvent être catégorisées de différentes manières. Il recommande plutôt d'utiliser des attributs dédiés pour étiqueter les catégories auxquelles les données appartiennent.

Exemple	Contre exemple
<pre><employees> <individu type="salarie" level="manager"> ... </individu> <individu type="directeur" level="manager"> ... </individu> <individu type="salarie" level="agent"> ... </individu> <individu type="contractuel" level="agent"> ... </individu> </employees></pre>	<pre><employees> <salarie> <manager> <individu .../> </manager> <agent> <individu .../> </agent> <directeur> <manager> <individu .../> </manager> </directeur> <contractuel> <agent> <individu .../> </agent> </contractuel> </employees></pre>

Le « *Marketplace* » est souvent contenu dans un « *Collection Element* ».

3.10.4.6 [Pattern-ReferencedNote] Il est recommandé d'isoler les annotations sur un contenu dans un élément séparé et les référencer à l'aide des attributs spéciaux « ID » et « IDREF ».

« *Referenced Note* » recommande d'isoler les annotations dans un élément séparé et de le référencer à l'aide des attributs spéciaux « ID » et « IDREF ».

Exemple	Contre exemple
<pre><paragraphe> Ceci est un texte annoté <reference ref="note"/>. </paragraphe> <note id="note">Une note.</note></pre>	<pre><paragraphe> Ceci est un texte annoté <note id="1">Une note.</note>. </paragraphe></pre>


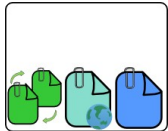

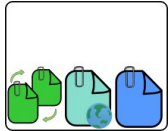
Ce *pattern* s'applique si un élément ou un texte nécessite d'être annoté, mais sans interrompre le flux du document. Si le document contient des passages narratifs annotés, cette règle s'applique très probablement.

3.10.4.7 **[Pattern-InformationGrouping]** L'utilisation de « *Information Grouping* » est recommandée dès lors qu'il s'agit de générer un document XML, à l'aide de XSL, à partir d'un autre.

« *Information Grouping* » recommande l'utilisation de transformation XSL, et décrit la stratégie à adopter, pour obtenir un document XML, produit à partir d'un autre document XML.

Ce *pattern* s'applique dès lors qu'un sous ensemble d'un document XML est nécessaire.

3.11 Performance

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[PERF-Compresser Document]	Si un document XML volumineux pose des problèmes d'échange sur le réseau, il doit être compressé.	<ul style="list-style-type: none"> Effective XML^[4] DMOD 	STD-DGFIP		
[PERF-AnalyseEvenementielDes DocumentsVolumineux]	Si un document XML est volumineux (plusieurs méga-octets), on doit l'analyser avec SAX.	<ul style="list-style-type: none"> Effective XML^[4] DMOD 	STD		

3.11.1 **[PERF-CompresserDocument]** Si un document XML volumineux pose des problèmes d'échange sur le réseau, il doit être compressé.

En effet, il est courant que les documents XML soient échangés sur le réseau, et, dans ce cas, la taille en octets du document peut engendrer un trafic important. Le réseau risque alors de devenir le goulot d'étranglement de l'applicatif (ou ce dernier peut simplement nuire aux performances des autres applicatifs utilisant le même réseau).

Dans ce genre de situation, une mauvaise pratique consiste à réduire la taille du document XML en simplifiant la syntaxe (transformer une balise « <nom> » en « <n> » par exemple). Ce n'est pas une bonne approche.

On ne doit pas chercher à réduire la taille d'un document XML en simplifiant la syntaxe des éléments. Cette technique nuit à la lisibilité du document mais surtout porte atteinte aux idées et objectifs fondamentaux de XML. Si la syntaxe est réduite, elle devient illisible et le document XML est finalement utilisé comme une sorte format binaire. C'est d'autant plus inefficace que le document XML résultant n'est pas performant comparé à un format binaire (il est trop verbeux). Le consommateur du flux aura plus de mal à l'analyser et devra probablement implémenter une série de traitements supplémentaires pour rendre leur sémantique aux données.

Si des problèmes de performance apparaissent lors des échanges de flux XML sur le réseau, il faut étudier la possibilité d'utiliser la compression comme solution technique. Cette compression peut se faire de manière applicative (au sein de l'applicatif) ou peut être fournie par un service technique (compression effectuée par l'instance d'Apache servant le flux par exemple).

Dans tous les cas, il est important de bien étudier l'impact de la compression sur le fonctionnement de l'application. En effet, dans certains cas, le coût CPU de la compression peut être prohibitif et entraîner de nouveaux problèmes de performance (cette fois-ci, sur le système hébergeant l'applicatif).

Si la compression ne permet pas de régler les problèmes de performance, il faut alors réfléchir sur la modélisation des données en elle même. N'y aurait-il pas un problème de conception ? Pourquoi transmettre tant d'informations sur le réseau ? Serait-il possible de transmettre moins de données (un sous ensemble du document initial par exemple) ?

Dans tout les cas, appauvrir la syntaxe XML, a des seules fins de performance, n'est pas la bonne solution.

3.11.2 [PERF-AnalyseEvenementielDesDocumentsVolumineux] Si un document XML est volumineux (plusieurs méga-octets), on doit l'analyser avec SAX.

Pour analyser un document XML, un *framework* dispose de 2 stratégies:






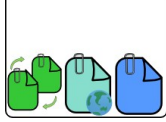


1. Charger l'intégralité du document en mémoire (DOM) ;
2. Parcourir le fichier et s'arrêter à certains noeuds, à la demande du programme appelant (SAX).

Du point de vue performance, surtout sur des fichiers de très gros volume, il est évident que l'approche SAX est la plus appropriée. Néanmoins, l'usage de DOM n'est pas proscrit pour les documents de petites tailles (< 1M). Si la taille du document est inconnue ou simplement variable, l'usage de SAX s'impose.

En outre, il est important de signaler que l'usage de SAX peut aussi être contre productif si le traitement effectué par l'appliquatif nécessite de nombreux « aller et retour » au sein du document pour corréler les informations.

3.12 Génération et publication de flux XML

Ces règles recouvrent les bonnes pratiques concernant les méthodes de génération de documents XML.

Réf.	Descriptif	Source(s)	Nature	Contrainte	Applicabilité
[GEN-PUB-Framework Xml]	La génération de document XML doit passer par un framework.	<ul style="list-style-type: none"> Effective XML^[4] DMOD 	STD-DGFIP		
[GEN-PUB-MediaType]	Un document XML doit être associé à un MIME TYPE approprié.	<ul style="list-style-type: none"> W3C DMOD 	STD-DGFIP		
[GEN-PUB-HttpEnc]	L'encodage ne doit pas être précisé dans la réponse HTTP.	<ul style="list-style-type: none"> W3C DMOD 	STD-DGFIP		
[GEN-PUB-ReferenceAccessible]	Les documents référencés par un document XML doivent être accessibles en même temps que le document principal.	<ul style="list-style-type: none"> DMOD 	STD-DGFIP		

3.12.1 [GEN-PUB-FrameworkXml] La génération de document XML **doit** passer par un framework.

Pour générer un flux XML, il est possible d'utiliser différents *frameworks* spécialisés dans la construction de documents XML. Une autre solution consiste à simplement « imprimer » le flux XML dans un fichier en utilisant les méthodes appropriées offertes par la machine virtuelle Java ou les autres technologies utilisées.

Les contre-exemples ci-dessous présentent les variantes proscrites :

Contre-Exemple en Java

```
System.out.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
System.out.println("<racine>");
List<Donnee> donnees = getDonnees();
for ( Donnee doneee : donnees ) {
    System.out.println("<donnee>" + doneee.toString() + "</donnee>")
}
System.out.println("</racine>");
```

Le même genre de procédé est utilisable dans un langage de script :

Contre-Exemple en Shell

```
echo '<?xml version="1.0" encoding="UTF-8"?>' >> ${OUTPUT}
echo '<racine>' >> ${OUTPUT}
...
echo '</racine>' >> ${OUTPUT}
```

Ces pratiques induisent plusieurs problèmes :

- la bonne formation du document XML n'est pas garantie ;
- l'intégrité de l'encodage choisi n'est pas garantie.

Explication :

1. Il n'est pas garanti que le flux XML soit bien formé : en effet, il appartient développeur de s'assurer que les balises générées par le programme sont bien formées. Si le fichier généré n'est pas consommé par le projet en lui même, il est tout à fait possible que le problème ne soit pas remonté immédiatement, et pose ensuite de graves complications en développement, en test d'intégration ou même en production. L'utilisation d'un parser XML évite ce genre de problème, laissant le développeur se concentrer sur la modélisation XML de ses données plutôt que sur la syntaxe du document.
2. Le respect de l'encodage n'est pas garanti : lors de ces différentes impressions, il est possible, et même facile, d'insérer par inadvertance un caractère n'appartenant pas à l'encodage choisi (UTF-8). Ainsi, le flux semblera valide mais sera vraisemblablement rejeté par certains « parseurs » XML qui ne pourront pas le consommer.

3.12.2 [GEN-PUB-MediaType] Un document XML **doit** être associé à un MIME TYPE approprié.

La RFC 3023 définit clairement les types MIME « application/xml » et « text/xml », et on autorise la création de sous types avec le suffixe +xml, comme par exemple "image/svg+xml".

Dans le cadre des projets de la DGFIP, un document XML doit être publié avec le type MIME « application/xml » et « text/xml ». L'utilisation de sous type spécifique est déconseillé pour des raisons d'interopérabilité.

De même, le type MIME « text/* » ne doit pas être utilisé pour servir des documents XML.

3.12.3 [GEN-PUB-HttpEnc] L'encodage **ne doit pas** être précisé dans la réponse HTTP.

La bonne pratique suivante du W3C a été retenue dans le cadre des projets DGFIP : « Le fournisseur d'une représentation ne devraient pas préciser l'encodage utilisé par le document XML dans les entêtes du protocole, car les données sont auto descriptives. »

De plus, dans le cadre des projets DGFIP, la règle [PROLOGUE-encodage] précise déjà que l'encodage doit être défini dans le prologue du document XML.

3.12.4 [GEN-PUB-ReferenceAccessible] Les documents référencés par un document XML **doivent** être accessibles en même temps que le document principal.

Les règles [PROLOGUE-Standalone] , [CONCEP-PasDeBinaire] et [CONCEP-DocumentIndépendant] incitent à rendre les documents les plus autonomes possible. Néanmoins, dans certains cas, comme par exemple les documents XML issus d'application tiers, il est nécessaire de publier des documents XML dépendant d'autres documents.

Dans ce contexte, il est nécessaire de s'assurer de la disponibilité de ces dépendances, de manière à exposer un document complet aux applicatifs utilisant le document.

4 Règles d'utilisation des espaces de noms XML


4.1 A propos de XML Namespace

La spécification *Namespaces in XML 1.0 (Second Edition)* décrit un mécanisme simple pour éviter les collisions de noms lors de la fusion de document XML. Ce mécanisme consiste simplement à attribuer à toute syntaxe XML un « espace de nom ».

Ainsi, en cas de fusion de plusieurs documents XML ayant des éléments aux identifiants identiques, les *frameworks* XML seront à même d'éviter toutes collisions. En outre, ils pourront clairement distinguer, dans le document résultant, l'origine de chaque élément. Pour identifier cet espace de nom, la spécification propose l'utilisation d'une URL.

Concrètement, l'espace de nom n'est donc qu'une URL à préciser, si besoin est, sur le premier élément d'un document XML.

4.2 Conventions de nommage

Réf.	Descriptif	Source(s)	Nature	Contrainte
[NAMESPACE-Version]	La partie chemin de l'URL de l'espace de noms doit être utilisée pour spécifier la version du document XML utilisée.	<ul style="list-style-type: none"> Effective XML^[4] 	STD-DGFIP	

[NAMESPACE-Version] La partie chemin de l'URL de l'espace de noms **doit** être utilisée pour spécifier la version du document XML utilisée.

Un document XML, comme tout document, est amené à évoluer et sa structure peut donc être changer. Pour permettre aux applicatifs utilisant le document de disposer d'une information sur sa version, on utilise l'espace de noms du document.

Celui ci étant une simple URL, il suffit d'ajouter au chemin la version du document.

Exemple pour la première version du projet « vespa »





`http://dgfip.finances.gouv.fr/vespa/1.0`

5 Règles associées à la validation XSD

5.1 Finalité, critères de sélection, sources et contraintes

Les schémas XML sont des documents XML dont la syntaxe permet de valider un document XML. Ils sont définis dans des fichiers XSD (XML Schema Description) et sont proposés comme mécanisme alternatif aux DTDs par le W3C. L'objectif des règles qui suivent est de normaliser autant que possible leur utilisation au sein des projets DGFIP, ceci principalement afin de faciliter leur lisibilité et leur maintenance.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSD-Obligatoire]	Un document XML doit être associé à une XSD	• DMOD	STD-DGFIP	
[XSD-PasDeDTD]	Un document XML doit utiliser une XSD et non une DTD pour valider un document XML	• DMOD	STD-DGFIP	
[XSD-EnteteFichier]	Un document XSD doit contenir un entête de fichier avec les informations minimales sur son propos, son rôle et sa maintenance.	• DMOD	STD-DGFIP	
[XSD-CamelCase]	Les noms des éléments ou des attributs doivent suivre la convention « camelCase », en commençant par une minuscule et en séparant les mots à l'aide de majuscules.	• DMOD • ADAE - Guide UML-XML ^[1e]	STD-DGFIP	
[XSD-Pattern-VenetianBlind]	La structuration d'un fichier XSD doit suivre le pattern « Venetian Blind ».	• ADAE - Guide UML-XML ^[1e] • MedBiquitous ^[4a]	STD-DGFIP	
[XSD-Pattern-PasBologno]	La structuration d'un fichier XSD ne doit pas suivre le pattern « Bologno ».	• MedBiquitous ^[4a]	STD-DGFIP	
[XSD-Namespace-Prefix]	Un schéma XSD doit utiliser le préfixe « xsd ».	• DMOD	STD-DGFIP	
[XSD-Namespace-EspaceDeNomParDefaut]	Un schéma XSD doit spécifier un espace de noms par défaut.	• DMOD • MedBiquitous	STD-DGFIP	
[XSD-NamespaceMêmeEspaceDeNom]	L'espace de noms d'un schéma XSD doit être le même que son espace de noms par défaut.	• MedBiquitous ^[4a]	STD-DGFIP	

<i>Réf.</i>	<i>Descriptif</i>	<i>Source(s)</i>	<i>Nature</i>	<i>Contrainte</i>
[XSD- Namespace- VersionSchema]	Un schéma XSD doit spécifier la version à la fin de l'URI de son espace de noms par défaut.	• MedBiquitous [4a]	STD-DGFIP	
[XSD- Namespace- ElementFormDefaultQualified]	Un schéma XSD doit définir l'attribut « elementFormDefault » à « qualified ».	• MedBiquitous [4a]	STD-DGFIP	
[XSD- Namespace- AttributeFormDefaultUnqualified]	Un schéma XSD doit définir l'attribut attributeFormDefault à unqualified.	• MedBiquitous [4a]	STD-DGFIP	
[XSD- Namespace- BlockDefaultAll]	Un schéma XSD doit définir l'attribut « blockDefault » à « #all ».	• MedBiquitous [4a]	STD-DGFIP	

5.2 Documents XML et XSD

5.2.1 [XSD-Obligatoire] Un document XML doit être associé à une XSD

Dans le contexte des projets DGFIP, tout document XML doit être non seulement bien formé mais aussi valide. Ceci rend indispensable la présence d'une XSD pour tout document XML.

En outre, comme le spécifie la règle [PROLOGUE-Standalone], il est important d'exposer des documents les plus indépendants et complets possible. Ainsi, lors de la publication ou de la diffusion d'un document XML, la XSD associée doit aussi être mise à disposition.

5.2.2 [XSD-PasDeDTD] Un document XML doit utiliser une XSD et non une DTD pour valider un document XML

Dans le cadre des projets DGFIP, le mécanisme de validation XML retenu est l'utilisation de schémas XML. L'utilisation de DTD est donc proscrite.

5.3 Un document XSD est avant tout un document XML

La plupart des règles associées aux documents XML s'applique donc également aux XSD.

5.3.1 Prologue, encodage et nommage

Les règles associées au prologue du document XML s'appliquent bien évidemment aux documents XSD :

- [PROLOGUE-XmlValid] ;
- [PROLOGUE-encodage] ;
- [PROLOGUE-XmlVersion] ;
- [PROLOGUE-Entete].

De même, l'ensemble des règles sur l'encodage ([XML-ENC-EntitesNecessaires] et [XML-ENC-FormeNormaleC]) s'applique également aux schémas XML.

Les règles de mise en forme suivantes s'appliquent aussi aux XSD.:

- [FORM-COM-EntiteAutorises] ;
- [FORM-COM-Encoding] ;
- [FORM-COM-PiedDeDocumentInterdit] ;
- [FORM-CDATA-PositionSection] ;
- [FORM-PRETTYPRINT-PresentationHierarchie] ;
- [FORM-PRETTYPRINT-PositionPremierAttribut].

Les règles de convention de nommage ([NOM-NomExplicite], [NOM-PrefixInterdit], et [NOM-UtilisationSymbole]) s'appliquent toutes aux XSD, à l'exception notable de [NOM-CamelCase] et [XML-EnteteFichier](cf. règles suivantes).

5.3.2 [XSD-EnteteFichier] Un document XSD **doit** contenir un entête de fichier avec les informations minimales sur son propos, son rôle et sa maintenance.

Tout fichier XSD doit contenir une description globale de son rôle et des informations de maintenance nécessaires aux outils classiques de gestion de configuration (CVS,SVN,...) indiquant le nom de l'auteur, la date de modification ainsi qu'une description sur la mise à jour (« logs »).

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright : Ministère du budget, des comptes publics et de la fonction publique - France
    Description de fichier
    $Id$
    $Log$
    Createur: Prénom Nom
    Date de création: Date
-->
<xsd:schema ...>
    (...)
</xsd:schema>
```

Note: Les valeurs telles que « \$Id\$ » ou « \$Log\$ » sont renseignées automatiquement par le serveur de gestion de configuration.

5.3.3 [XSD-CamelCase] Les noms des éléments ou des attributs **doivent** suivre la convention « camelCase », en commençant par une minuscule et en séparant les mots à l'aide de majuscules.

La mise en forme du contenu des documents XML respecte la règle [NOM-CamelCase], et reste en cohérence avec les conventions établies par le W3C.

5.3.4 Règles de conception XML applicable aux XSD

Les XSD disposent d'un ensemble de « Design Patterns » qui leur sont spécifiques.

Cependant, une partie des règles de conception XML ([CONCEP-QuandUtiliserCDATA], [CONCEP-DocumentIndépendant] et [CONCEP-ProfondeurArborescence]) s'applique aussi aux XSD.

De plus, de part leur sémantique définie par le W3C, les schémas XML ne peuvent pas enfreindre certaines règles de la présente charte ([CONCEP-PasDeBinaire], [CONCEP-UtiliseXHTMLPourLeContenuNarratif], [CONCEP-DonneeUnique], [CONCEP-NomElementPere], , [CONCEP-ConteneurListe], [CONCEP-ElementListe] , [CONCEP-ValeurAtomique] et [CONCEP-MaxAttributs]).

Dans le cas de la règle [CONCEP-MaxAttributs], il est à noter que bien qu'un schéma XML ne puisse pas enfreindre cette règle il doit en garantir le respect dans le document XML qu'il valide.

5.4 Structuration du document

Il existe plusieurs stratégies, clairement identifiées, pour structurer un document XSD. Ces stratégies sont désignées sous le terme « *XML Schema Patterns* ».

5.4.1 [XSD-Pattern-VenetianBlind] La structuration d'un fichier XSD **doit** suivre le pattern « Venetian Blind ».

Ce « *design pattern* », spécifique au XSD, présente les caractéristiques suivantes :

- un seul élément global, tous les autres sont donc locaux ;
- les sous éléments sont ajoutés en utilisant un type complexe ou les groupes d'éléments ;
- la réutilisation de ces types ou de ces groupes est possible seulement au sein du schéma.

Cette technique présente les avantages suivants :

- le schéma ne contient qu'un seul élément racine ;
- elle permet de réutiliser les types tout en ayant un seul élément global ;
- elle permet de diviser le schéma en plusieurs sous fichiers.

La seule limite est la faible encapsulation qui résulte de l'exposition des types.

Le choix de cette stratégie pour la construction du schéma XSD correspond à l'application du *pattern* XML « Universal Root ».

Exemple (« Venetian Blind »)

```
<xsd:schema
  blockDefault="#all"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.sun.com/point/venetianblind"
  xmlns:tns="http://schemas.sun.com/point/venetianblind"
  xmlns="http://schemas.sun.com/point/venetianblind"
  elementFormDefault="qualified">
  <!-- Declaration de types -->
  <xsd:complexType name="PointType">
    <xsd:attribute name="x" type="xsd:integer"/>
    <xsd:attribute name="y" type="xsd:integer"/>
  </xsd:complexType>
  <xsd:element name="Line"> <!-- Element racine -->
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PointA" type="PointType"/>
        <xsd:element name="PointB" type="PointType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

A titre de contre exemple, voici un fichier XSD qui suit le « pattern », non recommandé, « Salami Slice » (L'utilisation de ce *pattern* est une infraction à la règle [XSD-Pattern-VenetianBlind]) :

Contre exemple (« Salami Slice »)

```

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://dgi.spa/"
  elementFormDefault="qualified"
  xmlns:dgi="http://dgi.spa/">

  <!-- Validation du noeud 'modules' -->
  <xsd:element name="modules">
    <xsd:complexType>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="dgi:module"
            maxOccurs="unbounded"
            minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:complexType>
  </xsd:element>

  <!-- Validation du 'module' -->
  <xsd:element name="module">
    <xsd:complexType>
      <xsd:complexType>
        <xsd:element name="class"
          type="xsd:string"
          maxOccurs="1"
          minOccurs="1"/>
        <xsd:sequence>
          <xsd:element ref="params"
            maxOccurs="1"
            minOccurs="1"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:complexType>
    ...
  </xsd:element>
  ...
</xsd:schema>

```

5.4.2 [XSD-Pattern-PasBologno] La structuration d'un fichier XSD **ne doit pas** suivre le *pattern* « Bologno ».

Le *pattern* « Bologno » est en fait un mélange des différentes stratégies pour structurer un document XSD (« Venetian Blind », « Salami Slice »,...). Son utilisation est en contradiction avec la règle [XSD-Pattern-VenetianBlind], et indique une absence de stratégie dans la conception de la XSD. Par conséquence, son utilisation est fortement déconseillée.

5.5 Règles relatives aux Namespaces au sein des XSD

5.5.1 [XSD-Namespace-Prefix] Un schéma XSD **doit** utiliser le préfixe « xsd ».

Un schéma XSD **doit** utiliser le préfixe « xsd », et pas d'autres préfixes (tel que « xs » par exemple).

Exemple	Contre exemple
<pre><xsd:schema blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"></pre>	<pre><xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"></pre>

5.5.2 [XSD-Namespace-EspaceDeNomParDefaut] Un schéma XSD doit spécifier un espace de noms par défaut.

Pour assurer la lisibilité et la cohérence de l'ensemble des schémas XML de la DGFIP, un espace de noms par défaut associé à la syntaxe XSD doit être spécifié.

Exemple	Contre exemple
<pre><xsd:schema blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"></pre>	<pre><xsd:schema targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"></pre>

5.5.3 [XSD-NamespaceMêmeEspaceDeNom] L'espace de noms d'un schéma XSD doit être le même que son espace de noms par défaut.

Pour des raisons de clarté, de lisibilité et de conformité aux bonnes pratiques de l'industrie, l'espace de noms d'un schéma XSD doit être le même que son espace de noms par défaut.

Exemple
<pre><xsd:schema blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"></pre>

5.5.4 [XSD-Namespace-VersionSchema] Un schéma XSD doit spécifier la version à la fin de l'URI de son espace de noms par défaut.

Pour introduire la notion de version dans un document XSD, la pratique industrielle consiste à utiliser la partie chemin de l'URL utilisée pour l'espace de noms. On ajoute à la fin de l'URI l'information de version du schéma.

La valeur de l'espace de nommage par défaut étant la même que celle de l'espace de nommage, il est nécessaire d'ajouter cette information de version sur les deux URIs.

Exemple

```
<xsd:schema
  blockDefault="#all"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0"
  targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0"
  elementFormDefault="qualified">
```

5.5.5 [XSD-Namespace-ElementFormDefaultQualified] Un schéma XSD **doit** définir l'attribut « elementFormDefault » à « qualified ».

L'affectation de l'attribut « elementFormDefault » à « qualified » est une bonne pratique reconnue dans l'industrie. En effet, cette valeur impose d'utiliser les éléments au sein de leur espace de noms, de manière déclarative.

Cette pratique améliore la lisibilité du document, indiquant spécifiquement quels éléments appartiennent à quel espace de noms.

Exemple	Contre exemple
<pre><xsd:schema blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"> <xsd:element name="agents"> <xsd:complexType> <xsd:sequence> <xsd:element name="agent" type="xsd:string" maxOccurs="unbounded" /> </xsd:sequence> </xsd:complexType> </xsd:element> </xsd:schema></pre> <p>Exemple de document XML associé :</p> <pre><agents xmlns:role="http://dgfip.finances.gouv.fr/role"> <role:agent .../> </agents></pre>	<pre><xsd:schema blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="unqualified"> <xsd:element name="agents"> <xsd:complexType> <xsd:sequence> <xsd:element name="agent" type="xsd:string" maxOccurs="unbounded" /> </xsd:sequence> </xsd:complexType> </xsd:element> </xsd:schema></pre> <p>Exemple de document XML associé :</p> <pre><agents xmlns:role="http://dgfip.finances.gouv.fr/role"> <agent .../> </agents></pre>

Remarque : l'utilisation de cet attribut n'est pas sans conséquence pour les performances, car il force les *frameworks* à effectuer des opérations de validations plus poussées. En conséquence, cette règle peut être ignorée pour des raisons de performance.

5.5.6 [XSD-Namespace-AttributeFormDefaultUnqualified] Un schéma XSD doit définir l'attribut attributeFormDefault à unqualified.

Affecter la valeur « unqualified » à « attributeFormDefault » permet de ne pas utiliser le préfixe de l'espace de noms pour les attributs des éléments. En effet, préciser systématiquement le préfixe pour tout attribut aboutit à un document plus verbeux, tout en nuisant à sa lisibilité.

Exemple	Contre exemple
<pre><xsd:schema blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified" attributeFormDefault="unqualified"> ... </xsd:schema> Document XML associée : <tele-actes:formalite> ... <tele-actes:parties id="PP00001" qualite="D"></pre>	<pre><xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"> ... </xsd:schema> Document XML associée : <tele-actes:formalite> ... <tele-actes:parties tele-actes:id="PP00001" tele-actes:qualite="D"></pre>








5.5.7 [XSD-Namespace-BlockDefaultAll] Un schéma XSD doit définir l'attribut « blockDefault » à « #all ».

Cette attribut permet de définir la politique de gestion des dérivations. En affectant la valeur « #all » à l'attribut, on interdit explicitement toute forme de dérivation (substitution, extension et restriction). Le schéma XSD est donc « solidifié » et il devient impossible de modifier ses définitions en dérivant les types qu'il définit.

Exemple
<pre><xsd:schema blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://dgfip.finances.gouv.fr/refdoc/1.0" targetNamespace="http://dgfip.finances.gouv.fr/refdoc/1.0" elementFormDefault="qualified"></pre>

5.6 Règles de définition de type

Les schémas XML permettent de définir le type des éléments et attributs contenus dans un document XML. Ce chapitre décrit l'ensemble des règles à respecter dans la définition des types d'un document XML.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSD-Type-TypeDansNom]	Un schéma XSD ne doit pas définir les éléments en intégrant leur type à leur nom	• DMOD	STD-DGFIP	
[XSD-Type-TypeSimple]	Un schéma XSD doit utiliser les types simples.	• DMOD	STD-DGFIP	
[XSD-TYPE-PasDeRestrictionDeType]	Un schéma XSD ne doit utiliser la restriction que pour les types simples prédéfinis par XSD.	• DMOD • MedBiquitous ^[4a]	STD-DGFIP	
[XSD-TYPE-PasDeValeursParDefautNiFixe]	Il est recommandé qu'un schéma XML n'utilise pas les valeurs par défaut et les valeurs fixes	• ADAE - Guide UML-XML ^[1c]	STD-DGFIP	
[XSD-TYPE-PasDeRedefinitionDeType]	Le schéma XML ne doit pas utiliser de redéfinition de type.	• DMOD	STD-DGFIP	
[XSD-TYPE-PasDeSubstitution]	Un schéma XML ne doit pas utiliser les groupes de substitution.	• ADAE - Guide UML-XML ^[1c] • DMOD	STD-DGFIP	
[XSD-TYPE-NomTypeElementDifferent]	Au sein d'une XSD, on ne doit pas utiliser le même nom pour désigner un type et un élément.	• DMOD	STD-DGFIP	

5.6.1 [XSD-Type-TypeDansNom] Un schéma XSD **ne doit pas** définir les éléments en intégrant leur type à leur nom

Les schémas XSD permettent de définir les types des éléments. Ajouter le type au nom de l'élément n'est donc pas utile. Ainsi l'information sur le type ne sera pas redondante et ne polluera pas la sémantique du document XML en lui ajoutant des éléments sémantiques.

En outre, si le mainteneur du schéma oublie de modifier le type d'un élément dans son nom, cette information redondante peut facilement devenir une source d'erreur.

Ainsi, pour des raisons de simplicité, de maintenabilité et de lisibilité, dans un schéma XSD, le nom d'un élément ne doit pas contenir son type.

Exemple	Contre Exemple
<code><xsd:element name="Release" type="xsd:date" /></code>	<code><xsd:element name="ReleaseDate" type="xsd:date" /></code>

5.6.2 [XSD-Type-TypeSimple] Un schéma XSD doit utiliser les types simples.

La spécification des schémas XML définit un certain nombre de types pour représenter les données simples les plus courantes. Il est donc opportun d'utiliser autant que possible ces types prédéfinis.

Exemple	Contre Exemple
<code><xsd:element name="Release" type="xsd:date" /></code>	<pre> <xsd:element name="Release"> <xsd:sequence> <xsd:element name="annee" type="xsd:integer" /> <xsd:element name="mois" type="xsd:integer" /> <xsd:element name="jour" type="xsd:integer" /> </xsd:sequence> </xsd> </pre>

Ci dessous l'ensemble des différents types définis par la spécification des schémas XML :

- string
- boolean
- float
- double
- decimal
- precisionDecimal
- duration
- dateTime
- time
- date
- gYearMonth
- gYear
- gMonthDay
- gDay
- gMonth
- hexBinary
- base64Binary
- anyURI
- QName
- NOTATION

Remarque : cette règle n'interdit pas l'usage des type complexes (« `xsd:complexType` ») mais encourage l'utilisation des types prédéfinis lorsque cela est possible.

5.6.3 [XSD-TYPE-PasDeRestrictionDeType] Un schéma XSD ne **doit** utiliser la restriction que pour les types simples prédéfinis par XSD.

La restriction de type est très pratique pour limiter les valeurs possibles d'un type de données comme par exemple une chaîne de caractères ou une donnée numérique. Néanmoins, ce mécanisme peut aussi être utilisé pour modifier la définition d'un type défini par une autre XSD. L'utilisation de ce mécanisme pour assouplir ou modifier la description d'éléments est prohibée.

En outre, ce mécanisme n'est ni très simple, ni très explicite. Ainsi, l'utilisation d'autres mécanismes est fortement recommandée.

Exemple	Contre Exemple
<pre><xs:restriction base="xs:string"> <xs:pattern value="[0-9]{5}" /> </xs:restriction></pre>	<pre><xsd:restriction base="ChaineCaractereType"> <xsd:minLength value="13" /> <xsd:maxLength value="13" /> </xsd:restriction></pre>

5.6.4 [XSD-TYPE-PasDeValeursParDefautNiFixe] Il est recommandé qu'un schéma XML n'utilise pas les valeurs par défaut et les valeurs fixes.

L'utilisation de valeurs par défaut ou de valeurs fixes, définies au sein de la XSD, rend le document XML très dépendant de sa XSD. Bien qu'il soit fortement recommandé de diffuser les XSD avec le document XML([GEN-PUB-ReferenceAccessible]), il est impossible de le garantir.

Dans ce contexte, il est préférable d'assurer au document XML une indépendance complète vis à vis de sa XSD, et donc ne pas utiliser ce mécanisme.

5.6.5 [XSD-TYPE-PasDeRedefinitionDeType] Le schéma XML **ne doit pas** utiliser de redéfinition de type.

La spécification des schémas XML du W3C^[3c] autorise la redéfinition de type issue d'une autre XSD. Dans le contexte des projets DGFIP, ce mécanisme est prohibé.

Contre Exemple

```

<redefine
  schemaLocation="http://teleactes.real.not/v20081/teleactes_v20081.xsd">
  <xsd:complexType name="Extrait_M1">
    <xsd:sequence>
      <xsd:element name="numero" type="String1-12">
        <!-- Les 2 premiers caractères sont dans l'ordre suivant: un chiffre puis
une lettre.Les 10 caractères suivants sont obligatoirement des chiffres. Chaque
zéro doit être présent même si une suite de ce chiffre clôture ce numéro. -->
      </xsd:element>
      <xsd:element name="date" type="xsd:date">
        <!-- Date du modèle 1 -->
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</redefine>

```

5.6.6 [XSD-TYPE-PasDeSubstitution] Un schéma XML **ne doit pas** utiliser les groupes de substitution.

Le mécanisme de substitution de type est rendu impossible par l'utilisation du « *VenetianBlind* » ([XSD-Pattern-VenetianBlind]).

En outre, son utilisation va à l'encontre de la volonté, exprimée dans plusieurs règles ([XSD-TYPE-PasDeRedéfinitionDeType], [XSD-TYPE-PasDeRestrictionDeType]), de rendre les XSD les plus statiques possible.

En guise d'alternative, il est préférable d'utiliser d'autres mécanismes plus simples et plus faciles à maintenir.

Contre Exemple

```

<xs:element name="nom"
  type="xs:string"/>
<xs:element name="nomDeFamille"
  substitutionGroup="name"/>

```

On remarquera qu'il est possible d'interdire la substitution d'un élément avec l'ajout de l'attribut « `block="substitution"` ». Néanmoins, appliquer systématiquement à tout type d'élément cet attribut est fastidieux et risque de nuire à la lisibilité du schéma XSD.

5.6.7 [XSD-TYPE-NomTypeElementDifferent] Au sein d'une XSD, on **ne doit pas** utiliser le même nom pour désigner un type et un élément.



En effet, les types et les éléments ne sont pas dans le même espace de symboles, il est donc tout à fait possible de créer un type et un élément qui ont le même nom :

Exemple	Contre exemple
<pre> <xsd:complexType name="MatriculeFiscal"> ... </xsd:complexType> ... <xsd:element name="Matricule" type="MatriculeFiscal"/> </pre>	<pre> <xsd:complexType name="Matricule"> ... </xsd:complexType> ... <xsd:element name="Matricule" type="Matricule"/> </pre>

Cette pratique nuit à la lisibilité du schéma XML mais aussi rend plus complexe la réutilisation des types, puisqu'un type devient de facto associé à une instance d'élément précise.

5.7 Règles issues des « *patterns* » XML

Certains des *patterns* XML identifiés ont davantage de conséquences sur les schémas XSD que sur le document XML en tant que tel. Pour des soucis de lisibilité, les règles associées à ces *patterns* ont donc été placées dans ce chapitre.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSD-PATTERN-CatchAllElement]	Il est recommandé de structurer les sections du document pouvant contenir du XML inconnu.			
[XSD-PATTERN-CommonAttributes]	Il est recommandé de factoriser la définition des attributs communs à plusieurs éléments.			

5.7.1 [XSD-PATTERN-CatchAllElement] Il est recommandé de structurer les sections du document pouvant contenir du XML inconnu.

Parmi les « *patterns* » XML, le « *Catch-All Element* » recommande de définir ainsi les sections de document pouvant contenir du XML inconnu :

Exemple

```
<xsd:element name="car">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element
        name="model" type="string"/>
      <xsd:element
        name="year" type="string"/>
      <xsd:element name="comment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any
              namespace="http://www.w3.org/1999/xhtml"
              minOccurs="1" maxOccurs="unbounded"
              processContents="skip"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Ce *pattern* s'applique dès lors qu'un document contient des données qui peuvent être structurées dans un format XML inconnu.

5.7.2 [XSD-PATTERN-CommonAttributes] Il est recommandé de factoriser la définition des attributs communs à plusieurs éléments.

Parmi les *patterns* XML, le « *Common Attributes* » décrit comment implémenter, au sein d'une XSD, un ensemble d'attributs communs à plusieurs ou tous les éléments du document :

Exemple

```

<xsd:attributeGroup name="CommonAtts">
  <xsd:attribute name="id" type="ID"/>
  <xsd:attribute name="role" type="NMOKEN"/>
</xsd:attributeGroup>

<xsd:complexType name="NameType">
  <xsd:attributeGroup ref="CommonAtts"/>
</xsd:complexType>



<xsd:complexType name="PersonType"
  content="elementOnly">
  <xsd:element name="FirstName"
    type="NameType"/>
  <xsd:element name="LastName"
    type="NameType"/>
  <xsd:attributeGroup ref="CommonAtts"/>
</xsd:complexType>

<xsd:element name="Person"
  type="PersonType"/>

```

5.8 Règles de gestion des commentaires

La finalité des règles relatives aux commentaires de traitement est de faciliter la lisibilité de la XSD mais aussi sa maintenance par une documentation uniforme et adéquate du code.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSD-COM-UtiliserLesCommentairesXML]	Il est recommandé d'utiliser les commentaires XML pour documenter un schéma.	• DMOD	STD-DGFIP	
[XSD-COM-DocumenterAcronymes]	Si des acronymes sont utilisés pour les noms des types, ils doivent être documentés.	• ADAE - Guide UML-XML [1c] • DMOD	STD-DGFIP	

5.8.1 [XSD-COM-UtiliserLesCommentairesXML] Il est recommandé d'utiliser les commentaires XML pour documenter un schéma.

La spécification des schémas XML permet l'utilisation de l'usage des éléments « `xsd:documentation` » et « `xsd:annotation` » pour commenter une XSD. Un des intérêts de ces éléments est de permettre de définir la langue utilisée pour la documentation.

Dans le contexte des projets DGFIP, l'utilisation de cet élément n'est que peu pertinente. L'ensemble de la documentation des projets DGFIP doit être en français, ce qui rend l'attribut « `xml:lang="fr"` » redondant et alourdit inutilement le schéma.

En outre, la spécification des schémas XML n'interdit pas l'usage de simples commentaires XML au sein des XSD. En conséquence, pour éviter l'usage de différents mécanismes pour commenter les feuilles XSD, il est recommandé de n'utiliser que les commentaires XML.


Exemple	Contre exemple
<pre><xs:simpleType name="EnumQualiteIntervention"> <xs:restriction base="xs:string"> <xs:enumeration value="D"> <!-- Disposant (vendeur) --> </xs:enumeration> </xs:restriction> </xs:simpleType></pre>	<pre><xs:simpleType name="EnumQualiteIntervention"> <xs:restriction base="xs:string"> <xs:enumeration value="D"> <xs:annotation> <xs:documentation> Disposant (vendeur) </xs:documentation> </xs:annotation> </xs:enumeration> </xs:restriction> </xs:simpleType></pre>

5.8.2 [XSD-COM-DocumenterAcronymes] Si des acronymes sont utilisés pour les noms des types, ils **doivent** être documentés.

Les données manipulées par les projets STD-DGFIP englobent un large spectre d'acronymes. Il semble donc légitime de les utiliser pour définir des types de données XML. Néanmoins, les acronymes sont par essence obscurs pour les non initiés, il est donc important de bien documenter dans la XSD leur signification. Ce point est d'autant plus important que les intervenants d'un projet n'ont pas toujours la connaissance fonctionnelle de l'application.

5.9 Règles sur l'utilisation des PSVI

La spécification des schémas XML introduit la notion de « *Post Schema Validation Information* ». Elle autorise les validateurs XML à mettre à disposition de l'utilisateur un ensemble d'informations issues de l'analyse du schéma.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSD-PSVI-NePasUtiliserLesPSVI]	Un module applicatif ne doit pas utiliser les informations de post validation du schéma.	<ul style="list-style-type: none"> DMOD Effective XML^[4] 	STD-DGFIP	

[XSD-PSVI-NePasUtiliserLesPSVI] Un module applicatif **ne doit pas** utiliser les informations de post validation du schéma.

La spécification ne normalise en rien le format et la nature de ces éléments de « *post validation* ». De même, il n'est pas requis au *framework* XML de fournir ces informations. Dans ce contexte, il serait hasardeux de créer une couche supplémentaire de validation car ce mécanisme induirait une dépendance du traitement envers le validateur XML utilisé.

Par conséquent, l'utilisation des informations de « post validation » n'est pas autorisée dans le contexte des projets de la DGFIP.

6 XSL

XSL (*eXtensible Stylesheet Language*) est le langage de description de feuilles de style du W3C associé à XML. XSL a pour principal but la transformation et la mise en forme de document XML. Le langage regroupe trois spécifications :

- **XPath**, un langage de navigation dans un document XML, très fortement utilisé par XSLT ;
- **XSLT** (*XSL Transformation*), pour la transformation d'un document XML en divers formats tels que HTML ou encore XML (avec par exemple une autre arborescence de noeuds) ;
- **XSL-FO** (*XSL Formatting Object*), un langage pour la mise en forme et la mise en page de documents XML.

6.1 Règles relatives à XPath

6.1.1 Introduction

XPath est un langage de requête qui permet d'adresser et de désigner des objets contenus dans un document XML. Il est utilisé tant par XSLT que par d'autres technologies ne faisant pas l'objet de recommandations dans la présente charte (XPointer ou encore XML Query).

Dans XPath le référencement des noeuds s'effectue en décrivant des parcours dans l'arbre. Chaque parcours part d'un noeud *contexte* pour suivre un ou plusieurs *chemins* vers d'autres noeuds. Le langage d'adressage utilise donc ces chemins, pour désigner un ensemble d'objets. La désignation des chemins repose sur des notions d'axes et de sélection.

Plus précisément, un chemin, qui peut être absolu (commençant par « / ») ou relatif (absence du caractère initial « / » ou commençant par « . » pour indiquer le noeud courant), utilise des éléments de localisation (la recommandation parle d'étapes) qui se décomposent en :

- un axe, qui spécifie un sens de recherche, choisi parmi les attributs (*attribute*), les fils (*child*), les parents (*parent*), les ancêtres (*ancestor*) ou les frères (« *following-sibling* » ou « *preceding sibling* »), etc...
- un test ou filtre sur un type de noeud, par exemple, les ancêtres de type *chapitre* (« *ancestor::chapitre* ») ;
- un ou plusieurs prédicats, par exemple, le dernier des fils de type *chapitre* (« *child::chapitre[position()=Last()]* »).

Ces étapes se composent entre elles en utilisant l'opérateur `"/`. Par exemple, le titre du parent de type « *chapitre* » s'écrira « *ancestor::chapitre/titre* ».

La recommandation du W3C inclut une syntaxe abrégée pour XPath, allant directement à l'essentiel. Ainsi, « `//p` » sera une abréviation de « *self::node()/child::p* » : tous les éléments *p* contenus dans le sous-arbre en cours d'exploration (utilisation de l'axe *self*) et cela, quelle que soit leur position dans ce sous-arbre.

Enfin, pour définir les prédicats, XPath propose un langage d'expressions, utilisant un ensemble de fonctions de base, telles que les extractions de chaînes de caractères (« *concat*, *contains*, *start-with* », ...), les explorations de position d'un noeud parmi ses frères (« *position*, *last*, *count* », ...), etc.

En résumé, la syntaxe d'une expression XPath est constituée d'une suite d'étapes, séparées par des « / », comme illustré ci-dessous :

« `[/]`étape1/étape2/.../étapeN »

où le « / » initial est optionnel, permettant de distinguer un chemin absolu d'un chemin relatif. Les étapes sont évaluées de gauche à droite.

La forme générale d'une étape est donc :

« `axe::filtre[prédicat1][prédicat2]...` »

L'évaluation d'une étape peut être soit une valeur, soit un ensemble de noeuds (*node-set*). Notons qu'un noeud n'est pas extrait du document, ni placé hors de sa position dans l'arbre. Un *node-set* retourné par une évaluation, devrait être plutôt considéré comme un ensemble de références vers des noeuds de l'arbre XML. L'obtention de cet ensemble se fait toujours par rapport à un *noeud contexte* ou *noeud contextuel* à partir duquel une évaluation est effectuée. Ainsi dans une expression incluant plusieurs étapes, le noeud contexte à partir duquel on calcule un chemin de localisation va changer.




Dans une expression XPath comme celle-ci « `/A/B/@attrib` », et lors de la première étape, les éléments de type « A » sont recherchés; on obtient une liste à partir duquel on prend pas à pas chacun de ses éléments pour retrouver les éléments « B », et on fait de même pour les éléments de type « B » pour retrouver l'attribut « `attrib` ».

Chaque étape retourne donc une liste d'éléments (du type *node*, *attribute*, ...), et chacun de ces éléments dispose d'une position dans cette liste. Des fonctions sont fournies par XPath pour déterminer la position d'un élément dans un ensemble : « `position()` », ou encore « `last()` » qui retourne la taille de l'ensemble (qui est en fait la dernière position).

XPath est fortement utilisé par XSLT. Dans le cadre de cette charte, ne seront considérées comme règles pour XPath que celles qui sont pertinentes pour XSLT.

6.1.2 Finalité, critères de sélection, sources et contraintes

L'ensemble des règles décrites dans cette section porte sur des expressions XPath utilisables dans des feuilles de styles XSLT.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XPATH-SlashSlash]	L'utilisation de « <code>//</code> » n'est pas recommandée dans les expressions XPath.	• John E. Simpson ^[4d]	STD-DGFIP	
[XPATH-EnfantImmédiat]	On ne doit pas utiliser une expression XPath qui commence avec « <code>//</code> » « <code>/</code> » ou « <code>..</code> » si des enfants immédiats d'un noeud sont recherchés.	• John E. Simpson ^[4d]	STD-DGFIP	
[XPATH-FiltreAttribut]	Un filtrage par attribut ne doit intervenir que dans la dernière étape d'une expression Xpath.	• W3C	NOR	

6.1.2.1 [XPATH-SlashSlash] L'utilisation de « `//` » n'est pas recommandée dans les expressions XPath.

L'opérateur « `//` » est un raccourci pour l'expression « `/descendant-or-self::node()` », qui recherche tous les descendants d'un noeud dans un document, et pas seulement les enfants immédiats. Quand elle est utilisée au début d'une expression XPath, elle entraîne une recherche récursive depuis la racine du document. Ce mécanisme peut devenir très coûteux, en terme de performance, surtout pour des documents volumineux.

Considérons le document XML suivant:

Document XML d'exemple

```

<Personne>
  <coordonnes>
    <adresse>
      <num>...</num>
      <rue>...</rue>
      <ville>...</ville>
      ...
    </adresse>
    <telephones>
      <domicile>01...</domicile>
      <mobile>06...</mobile>
    </telephones>
  </coordonnes>
</Personne>

```

Si l'on souhaite obtenir les numéros de téléphone mobile, on pourra utiliser l'une des deux expressions suivantes :

Exemple	Contre exemple
/Personne/telephones/mobile	//mobile

L'expression « /Personne/telephones/mobile » permet d'atteindre l'élément « mobile » plus rapidement et surtout uniquement si celui-ci est le fils d'un élément « telephones », lui même fils de « Personne ».

A l'inverse, l'expression « //mobile » va parcourir tous les descendants depuis la racine, incluant des détours inutiles par les éléments « adresse » ou « domicile ».

6.1.2.2 [XPATH-EnfantImmédiat] On **ne doit pas** utiliser une expression XPath qui commence avec « // » « / » ou « .. » si des enfants immédiats d'un noeud sont recherchés.

Comme vu dans la règle précédente, au début d'une expression, « // » retourne les descendants, « / » recherche depuis la racine du document alors que « .. » remonte au noeud parent et recherche à partir de ce noeud.

En considérant le même code source que celui de la règle précédente, si l'on souhaite rechercher un numéro de « mobile » depuis le noeud « telephones », l'exemple suivant montre qu'il est plus judicieux d'utiliser directement la syntaxe « mobile ».

Exemple	Contre exemple
mobile	<pre> //mobile recherche tous les noeuds mobile depuis le noeud telephones ../mobile recherche les noeuds mobile fils du parent /mobile recherche les noeuds mobile depuis la racine </pre>

6.1.2.3 [XPath-FiltreAttribut] Un filtrage par attribut **ne doit intervenir que** dans la dernière étape d'une expression XPath.

D'un point de vue XPath, un attribut est vu comme un noeud feuille. Par conséquent il ne peut apparaître que dans une dernière étape.

Exemple	Contre exemple
/A/B/@attrib1	/A/B/@attrib1/C

6.2 Règles relatives à XSLT

6.2.1 Introduction

La transformation d'un document XML en un autre document requiert l'utilisation d'un processeur XSLT. Ce dernier reçoit en entrée le document XML, ainsi que les instructions de transformation à appliquer.

Ces instructions ou règles sont décrites dans un document XSLT, c'est-à-dire une feuille de styles XSLT. Chaque règle possède un motif (*pattern*) ainsi qu'un modèle (*template*). Le processeur XSLT parcourt et compare les éléments (noeuds, attributs, ...) du document XML en entrée avec les règles de transformation présentes dans une feuille de styles. Lorsqu'il y a correspondance, il écrit le modèle de cette règle dans le document de destination, pouvant être un autre document XML ou toute autre format, comme du texte brut ou du HTML.

Il existe plusieurs processeurs XSLT, dont certains peuvent être intégrés dans un navigateur Web, un serveur Web ou un serveur applicatif. Certains processeurs peuvent aussi être exécutés en ligne de commande. Une liste assez complète des processeurs XSLT peut être obtenue sur le site <http://xml.coverpages.org/xslSoftware.html>, et parmi les plus utilisés dans le domaine open source, on notera Xalan (<http://xml.apache.org/xalan-j/>) et Saxon (<http://saxon.sourceforge.net/>).

XPath est fortement utilisé par XSLT comme langage de référencement. Les cas d'utilisation sont listés ci-dessous :

- sélection de noeuds auxquels on souhaite appliquer une règle : dans ce cas l'expression XPath apparaît dans l'attribut *select* d'une instruction de type *xsl-apply-templates* ;
exemple : `<xsl:apply-templates select="titre/chapitre"/>`
- désignation des noeuds auxquels s'applique une règle : dans ce cas l'expression XPath apparaît dans l'attribut *match* ;
exemple :
`<xsl:template match="/">`
 .
 .
 .
`<xsl:template>`
- extraction de valeurs:
exemple : `<xsl:value-of select="livre/@auteur">`
- prédicats de test.
exemple : `<xsl:if test="($titre= 'Harry Potter') ">`

6.2.2 Règles XML applicables à un document XSLT

6.2.2.1 Prologue, encodage et nommage

Les règles associées au prologue du document XML s'appliquent bien évidemment aux documents XSLT :

- [PROLOGUE-XmlValid] ;
- [PROLOGUE-encodage] ;
- [PROLOGUE-XmlVersion] ;
- [PROLOGUE-Entete].

De même, l'ensemble des règles sur l'encodage ([XML-ENC-EntitesNecessaires] et [XML-ENC-FormeNormaleC]) s'applique également aux documents XSLT.

Les règles de mise en forme suivantes s'appliquent aussi aux documents XSLT :

- [FORM-COM-EntiteAutorises] ;
- [FORM-COM-Encoding] ;
- [FORM-COM-PiedDeDocumentInterdit] ;
- [FORM-CDATA-PositionSection] ;
- [FORM-PRETTYPRINT-PresentationHierarchie] ;
- [FORM-PRETTYPRINT-PositionPremierAttribut].

Les règles de convention de nommage ([NOM-NomExplicite], [NOM-PrefixInterdit], et [NOM-UtilisationSymbole]) s'appliquent toutes aux documents XSLT, à l'exception notable de [NOM-CamelCase] (cf. règle [XSLT-CamelCase]).

6.2.3 Règles de conception XML applicables à un document XSLT

Les règles de conception XML ([CONCEP-QuandUtiliserCDATA] et [CONCEP-ProfondeurArborescence]) s'appliquent aussi aux documents XSLT. A l'inverse, la règle [CONCEP-DocumentIndépendant] ne s'applique pas aux documents XSLT.

Le contexte des documents XSLT étant différent d'un simple document XML, les règles suivantes ne peuvent simplement pas s'appliquer :





- [CONCEP-PasDeBinaire] ;
- [CONCEP-UtiliseXHTMLPourLeContenuNarratif] ;
- [CONCEP-DonneeUnique] ;
- [CONCEP-NomElementPere] ;
- [CONCEP-ConteneurListe] ;
- [CONCEP-ElementListe] ;
- [CONCEP-ValeurAtomique].

Les documents XSLT disposent d'un ensemble de « *Design Patterns* » qui leurs sont spécifiques.

6.2.4 Règles sur la présentation d'un document XSLT

6.2.4.1 Finalité, critères de sélection, sources et contraintes

Les règles de présentation décrites ci-dessous ont pour but d'améliorer la lisibilité et la compréhension d'une feuille de style.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-CamelCase]	Les noms des éléments ou des attributs doivent être écrits en minuscules, en séparant les mots à l'aide de traits d'union.	• W3C	NOR	
[XSLT-NomSuffix]	Un document XSLT doit avoir le suffixe « .xsl ».	• DMOD	STD-DGFIP	
[XSLT-Nom]	Un document XSLT doit avoir un nom qui rend compte de son rôle.	• DMOD	STD-DGFIP	
[XSLT-NomCorrect]	Un document XSLT ne doit pas comporter de caractères accentués.	• DMOD	STD-DGFIP	

6.2.5 [XSLT-CamelCase] Les noms des éléments ou des attributs **doivent** être écrits en minuscules, en séparant les mots à l'aide de traits d'union.

Cette syntaxe diffère de la règle [NOM-CamelCase] pour suivre les conventions établies par le W3C.

6.2.6 [XSLT-NomSuffix] Un document XSLT **doit** avoir le suffixe « .xsl ».

De manière générale, un document XSLT est reconnu par la présence des éléments XML appropriés dans son entête. Néanmoins, le suffixe utilisé pour son nommage devrait aussi donner une indication sur sa nature. Pour des raisons d'harmonisation, et en cohérence avec les pratiques de l'industrie, il est préconisé d'utiliser l'extension « .xsl ».

Exemple	Contre exemple
matransformation.xsl	matransformation.xml

6.2.6.1 [XSLT-Nom] Un document XSLT **doit** avoir un nom qui rend compte de son rôle.

Le nom d'un document XSLT doit refléter sa nature ainsi que son rôle via son nom.

Exemple	Contre exemple
fromDocbookToHtml.xsl	stylesheet.xsl

6.2.6.2 [XSLT-NomCorrect] Un document XSLT **ne doit pas** comporter de caractères accentués.




Les caractères accentués ne doivent pas être utilisés pour définir le nom d'un fichier XSLT.

Exemple	Contre exemple
conversionFichierEmployesVersHtml.xml	conversionFichierEmployésVersHtml.xml

6.2.7 Règles de documentation

6.2.7.1 Finalité, critères de sélection, sources et contraintes

Les règles décrites ci-dessous ont pour but d'améliorer la documentation et la compréhension d'une feuille de style. Rappelons que les commentaires rédigés dans les feuilles de styles doivent être en français, et suivent évidemment les règles syntaxiques définies par la norme XML, à savoir une encapsulation par les balises « <!-- » et « --> ».

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-EnteteFichier]	Un document XSLT doit contenir un entête de fichier avec informations minimales sur son propos, son rôle et son fonctionnement.	• DMOD	STD-DGFIP	
[XSLT-EnteteRegle]	Une règle XSLT doit être commentée afin de comprendre son fonctionnement.	• DMOD	STD-DGFIP	
[XSLT-VersionDocument]	Un document XSLT doit avoir une variable « version », placée comme premier fils de « stylesheet », qui contient le numéro de version du document.	• DMOD • Effective XML ^[4]	STD-DGFIP	

6.2.7.2 [XSLT-EnteteFichier] Un document XSLT **doit** contenir un entête de fichier avec informations minimales sur son propos, son rôle et son fonctionnement.

Tout fichier XSLT doit contenir une description globale de son rôle et des informations de maintenance nécessaires aux outils classiques de gestion de configuration (CVS,SVN,...) indiquant le nom de l'auteur, la date de modification ainsi qu'une description sur la mise à jour (« logs »).

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright : Ministère du budget, des comptes publics et de la fonction publique - France
    Description de fichier
    $Id$
    $Log$
    Createur: Prénom Nom
    Date de création: Date
-->
<xsl:stylesheet version="1.0" xmlns:xsl="URI">
    (...)
</xsl:stylesheet>
```

Note: Les valeurs telles que ou « \$Id\$ » ou « \$Log\$ » sont renseignés automatiquement par le serveur de gestion de configuration.

6.2.7.3 [XSLT-EnteteRegle] Une règle XSLT **doit** être commentée afin de comprendre son fonctionnement.

Toute règle doit être documentée afin d'en comprendre le sens, de manière à faciliter sa maintenance.

Exemple**Contre exemple**

```
<!-- Description de la règle -->
<xsl:template ...>
    . . .
</xsl:template>
```

```
<xsl:template ...>
    . . .
</xsl:template>
```

6.2.7.4 [XSLT-VersionDocument] Un document XSLT **doit** avoir une variable « version », placée comme premier fils de « stylesheet », qui contient le numéro de version du document.

A la différence de la plupart des documents XML, les documents XSLT sont des scripts et il est important de les versionner de la même manière que n'importe quel autre fichier source. La pratique veut que l'on utilise la partie chemin de l'URL de l'espace de noms pour indiquer la version d'un document XML. Dans le cas, d'un document XSLT, ce chemin est normalisé et contient la version du langage XSLT.

Ainsi, l'industrie a adopté la pratique de placer les informations relatives à la version d'un document XSLT dans une variable « version », placée sous l'élément racine, « stylesheet ».

En outre, cette pratique permet de disposer de cette information au sein du script, ce qui peut être pratique pour générer, par exemple, des messages d'erreurs détaillés.

Exemple





```
<xsl:stylesheet version="1.0"
  xmlns:xsl="URI">
  <xsl:variable name=version
    value="'2.1'"/>
  ...
  <xsl:message>
    Un problème est survenu lors de l'exécution de la version <xsl:value-of select="$version"/>
    de ce document.
  </xsl:message>
</xsl:stylesheet>
```

On note aussi que cette règle n'est pas en contradiction avec la règle [XSLT-Version] qui indique simplement la manière de déclarer la version de XSLT utilisée.

6.2.8 Règles sur la Structure d'un document XSLT

6.2.8.1 Finalité, critères de sélection, sources et contraintes

Cette section décrit les règles à respecter pour définir la structure d'un document XSLT.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-Racine]	Un document XSLT doit débuter par l'élément « stylesheet »	• W3C	NOR	
[XSLT-Version]	L'élément XSLT racine doit contenir la version XSLT 1.0.	• W3C	NOR	
[XSLT-NameSpace-Prefix]	Un document XSLT doit définir le préfixe « xsl: »	• W3C	NOR	
[XSLT-NomsElements]	Les noms des éléments XSLT, des attributs et des fonctions doivent être en lettres minuscules, et utiliser le trait d'union pour séparer les mots.	• W3C	STD-DGFIP	

6.2.8.2 [XSLT-Racine] Un document XSLT **doit** débuter par l'élément « *stylesheet* »

Une feuille de styles est représentée par l'élément « `xsl:stylesheet` » dans un document XML.

Exemple

Contre exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="nombre"
  xmlns:xsl="URI">
  (...)
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="nombre"
  xmlns:xsl="URI">
  (...)
</xsl:transform>
```

6.2.8.3 [XSLT-Version] L'élément XSLT racine **doit** contenir la version XSLT 1.0.

La version de XSLT retenue pour les projets DGFIP est la version 1.0, la version 2.0, dernière en date, n'étant pas encore adoptée largement par l'industrie. Par souci de clarté, et pour assurer la bonne interprétation de la feuille de style par tout *framework*, tout document XSLT doit spécifier la version du langage utilisée.

Exemple	Contre exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="..."> (...) </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="..."> (...) </xsl:stylesheet></pre>

6.2.8.4 [XSLT-NameSpace-Prefix] Un document XSLT **doit** définir le préfixe « xsl: »

Comme pour tout document XML, il est impossible de préfixer les éléments de l'espace de noms XSLT par l'acronyme de son choix. Pour souci de clarté et d'homogénéisation, le préfixe « xsl: » est le seul retenu pour définir l'espace de noms XSL.

Remarque: L'URI associé à cet espace de noms est :
« <http://www.w3.org/1999/XSL/Transform> ».

Exemple	Contre exemple
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/ Transform"> (...) </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <xslt:stylesheet version="1.0" xmlns:xslt="http://www.w3.org/1999/XSL/Tra nsform"> (...) </xslt:stylesheet></pre>

6.2.8.5 [XSLT-NomsElements] Les noms des éléments XSLT, des attributs et des fonctions **doivent** être en lettres minuscules, et utiliser le trait d'union pour séparer les mots.

Les conventions utilisées pour les noms des éléments XSLT, des attributs et des fonctions diffèrent de celles retenues par la DGFIP (cf. [NOM-CamelCase]). Néanmoins, par souci de cohérence avec les bonnes pratiques de l'industrie et les conventions retenues par le W3C, la convention retenue est donc :




- tous les noms sont écrits en lettres minuscules ;
- le trait d'union est utilisé pour séparer les mots.

Exemple	Contre exemple
<pre><!-- Element XSLT --> <xsl:for-each .../> <!-- Fonction --> string-length()</pre>	<pre><!-- Element XSLT --> <xsl:forEach ...> <!-- Fonction --> stringLenght()</pre>

6.2.9 Règles sur les éléments de premier niveau

6.2.9.1 Finalité, critères de sélection, sources et contraintes

Les types d'éléments qui peuvent être fils de l'élément racine `<xsl:stylesheet>` sont dits « éléments de premier niveau » ou de « haut niveau ». Les règles décrites ci-dessous ont pour but d'améliorer leur conformité au standard.

<i>Réf.</i>	<i>Descriptif</i>	<i>Source(s)</i>	<i>Nature</i>	<i>Contrainte</i>
[XSLT-PremierNiveau]	Les éléments de haut niveau doivent être du type standard et conformes à l'espace de noms XSLT.	• W3C	NOR	
[XSLT-TextPremierNiveauInterdit]	Il ne doit pas y avoir des noeuds de type texte placés immédiatement sous l'élément racine.	• W3C	NOR	
[XSLT-PrefixeProcesseur]	Les extensions spécifiques au processeur XSLT ne doivent pas être utilisées.	• DMOD	STD-DGFIP	

6.2.9.2 [XSLT-PremierNiveau] Les éléments de haut niveau doivent être du type standard et conformes à l'espace de noms XSLT.

Les éléments de premier niveau sont définis par l'espace de noms XSLT. Les éléments de premier niveau doivent être conformes à l'un des types suivants :

- `xsl:import ;`
- `xsl:include ;`
- `xsl:strip-space ;`
- `xsl:preserve-space ;`
- `xsl:output ;`
- `xsl:key ;`
- `xsl:decimal-format ;`
- `xsl:namespace-alias ;`
- `xsl:attribute-set ;`
- `xsl:variable ;`
- `xsl:param ;`
- `xsl:template.`

L'ordre d'apparition de ces éléments n'est pas important sauf pour « `xsl:import` », qui doit apparaître en premier s'il est utilisé.

Exemple

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="..." />
  <xsl:include href="..." />
  <xsl:strip-space elements="..." />
  <xsl:preserve-space elements="..." />
  <xsl:output method="..." />
  <xsl:key name="..." match="..." use="..." />
  <xsl:decimal-format name="..." />
  <xsl:namespace-alias stylesheet-prefix="..." result-prefix="..." />
  <xsl:attribute-set name="...">
    ...
  </xsl:attribute-set>
  <xsl:variable name="...">...</xsl:variable>
  <xsl:param name="...">...</xsl:param>
  <xsl:template match="...">
    ...
  </xsl:template>
  <xsl:template name="...">
    ...
  </xsl:template>
</xsl:stylesheet>
```

6.2.9.3 [XSLT-TextPremierNiveauInterdit] Il ne doit pas y avoir des noeuds de type texte placés immédiatement sous l'élément racine.

Il est interdit d'avoir des noeuds de type texte, placés immédiatement sous l'élément racine. On peut en revanche trouver des commentaires qui sont ignorés.

Exemple	Contre-Exemple
<pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <!-- Ceci est un exemple de texte autorisé comme commentaire --> </xsl:stylesheet></pre>	<pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> Ceci est un exemple de texte à ne pas inclure dans sortie </xsl:stylesheet></pre>
<pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl-template match="/"> <h1> Bonjour </h1> </xsl-template> </xsl:stylesheet></pre>	<pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <h1> Bonjour </h1> </xsl:stylesheet></pre>

6.2.9.4 [XSLT-PrefixeProcesseur] Les extensions spécifiques au processeur XSLT ne doivent pas être utilisées.

Les éléments spécifiques au processeur XSLT, qui se distinguent en général par des préfixes comme « xalan: » ou « saxon: », doivent être évités, car leur introduction poserait des problèmes de portabilité.

Contre-Exemple
<pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <saxon:output ...> </saxon:output> </xsl:stylesheet></pre>
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xalan="http://xml.apache.org/xalan"> <xsl:output method="xml" encoding="UTF-8" indent="yes" xalan:indent-amount="2"/></pre>






6.2.10 Modularité et inclusion de feuille de styles




6.2.10.1 Finalité, critères de sélection, sources et contraintes

Une feuille de styles XSLT peut consister en un ou plusieurs modules XSLT, définis dans des fichiers séparés. XSLT fournit deux mécanismes pour combiner des feuilles de styles :

- « *xsl:include* », est un mécanisme d'inclusion permettant aux feuilles de styles d'être combinées sans changer la sémantique des feuilles de styles à combiner ;
- « *xsl:import* », un mécanisme d'import permettant aux feuilles de styles de se recouvrir mutuellement.

Les règles décrites ci-dessous ont pour but de définir leur bon usage et le périmètre de leur utilisation.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-UtiliserInclude]	L'inclusion de feuilles de styles doit être réalisée à l'aide de l'instruction « <i>xsl:include</i> ».	• DMOD	STD-DGFIP	
[XSLT-ImportAvecMode]	Il est recommandé de redéfinir les règles, si besoin est, à l'aide de l'attribut « <i>mode</i> »	• W3C	NOR	
[XSLT-Include]	L'élément « <i>xsl:include</i> » ne doit être utilisé que comme élément de premier niveau (fils de l'élément « <i>xsl:stylesheet</i> »).	• W3C	NOR	
[XSLT-AutoInclude]	Une feuille XSLT ne doit pas s'inclure elle-même (directement ou indirectement).	• W3C	NOR	
[XSLT-IncludeUneFois]	Un fichier <i>xsl</i> ne doit être inclus qu'une seule fois.	• W3C	NOR	

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-Import]	On ne doit utiliser l'élément « xsl:import » que comme élément de premier niveau, et il doit précéder tous les autres éléments fils de « xml:stylesheet ».	• W3C	NOR	
[XSLT-AutoImport]	Une feuille XSLT ne doit pas s'importer elle-même (directement ou indirectement).	• W3C	NOR	
[XSLT-ImportUneFois]	Un fichier xsl ne doit être importé qu'une fois avec xsl:import.	• W3C	NOR	

6.2.10.2 [XSLT-UtiliserInclude] L'inclusion de feuilles de styles **doit** être réalisée à l'aide de l'instruction « xsl:include ».

Si une feuille de style utilise des règles définies dans une autre feuille de style, elle doit utiliser l'instruction « xsl:include ». En effet, cette instruction interdit de redéfinir, par inadvertance ou par choix, les règles importées.

A l'inverse, l'usage de « xsl:import », plus permissif, doit être justifié et les règles s'appliquant sur ce type d'élément XSL doivent être respectées.

6.2.10.3 [XSLT-ImportAvecMode] Il est recommandé de redéfinir les règles, si besoin est, à l'aide de l'attribut « mode ».

Si l'on souhaite redéfinir une règle importée ou incluse ou bien simplement définir plusieurs comportements possibles sur un seul élément, il est recommandé d'utiliser l'attribut « mode », qui permet d'obtenir de manière lisible ce comportement.

Ce mécanisme permet donc de redéfinir, de manière explicite, une règle, sans avoir recours à l'utilisation d'import (« xsl:import »).

L'attribut mode permet à un élément d'avoir plusieurs modèles, il ne peut être utilisé en l'absence de l'attribut match.

Exemple

Feuille1.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
  version = "1.0" >
  ...
  <xsl:template match="employe" mode="calcul-de-salaire">
    ...
  </xsl:template>
</xsl:stylesheet>
```

Feuille2.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
  version = "1.0" >
  <xsl:import href="feuille1.xsl"/>
  ...
  <xsl:template match="employe" mode="calcul-point-retraite">
    ...
  </xsl:template>

  <xsl:template match="/">
    ...
    <!-- Génération de la partie salaire de la fiche de l'employé -->
    <xsl-apply-templates select="test" mode="calcul-de-salaire"/>
    <!-- Génération de la partie retraite de la fiche de l'employé -->
    <xsl-apply-templates select="test" mode="calcul-point-retraite"/>
    ...
  </xsl:template>
</xsl:stylesheet>
```

6.2.10.4 [XSLT-Include] L'élément « `xsl:include` » **ne doit être** utilisé que comme élément de premier niveau (fils de l'élément « `xsl:stylesheet` »).

L'élément « `xsl:include` » ne doit apparaître que comme élément de haut niveau.

Exemple

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:include href="xsltest.xsl"/>
  . . .
  <xsl:template match="/" >
    <xsl:apply-templates select="//AA"/>
  </xsl:template>
</xsl:stylesheet>
```

Contre-Exemple

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  . . .
  <xsl:template match="/" >
    <xsl:include href="xsltest.xsl"/>
    <xsl:apply-templates select="//AA"/>
  </xsl:template>
</xsl:stylesheet>
```

6.2.10.5 [XSLT-AutoInclude] Une feuille XSLT **ne doit pas** s'inclure elle-même (directement ou indirectement).

Le comportement d'une inclusion en « boucle » est variable selon les *frameworks* de transformation XSLT. En outre, ce genre de pratique nuit à la compréhension, déjà peu aisée, des feuilles de styles.

Pour toutes ces raisons, une feuille de styles ne doit pas s'inclure elle-même.

Contre-Exemple

```
<!-- fichierA.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  . . .
  <xsl:include href="fichierA.xsl"/>
  . . .
</xsl:stylesheet>

<!-- fichierB.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  . . .
  <xsl:include href="fichierA.xsl"/>
  . . .
</xsl:stylesheet>

<!-- fichierA.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  . . .
  <xsl:include href="fichierB.xsl"/>
  . . .
</xsl:stylesheet>
```

6.2.10.6 [XSLT-IncludeUneFois] Un fichier xsl ne doit être inclus qu'une seule fois.

Une inclusion multiple d'un même fichier XSLT peut provoquer des erreurs lors de l'exécution.

Note: Les inclusions multiples sont moins évidentes lorsqu'elles sont indirectes. Comme dans le contre-exemple suivant où l'on constate que le fichier « D » inclut deux fois le fichier « A.xsl ».

Contre-Exemple
<pre> <!-- Fichier B.xml--> <xsl:stylesheet version="1.0" ..."> <xsl:include href="A.xsl"/> . . . </xsl:stylesheet> <!-- Fichier C.xml--> <xsl:stylesheet version="1.0" ..."> <xsl:include href="A.xsl"/> . . . </xsl:stylesheet> <!-- Fichier D.xml--> <xsl:stylesheet version="1.0" ..."> <xsl:include href="B.xsl"/> <xsl:include href="C.xsl"/> . . . </xsl:stylesheet> </pre>

6.2.10.7 [XSLT-Import] On ne doit utiliser l'élément « xsl:import » que comme élément de premier niveau, et il doit précéder tous les autres éléments fils de « xml:stylesheet ».

L'élément « xsl:import » ne doit apparaître que comme élément de haut niveau.

Exemple	Contre-Exemple
<pre> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:import href="xslttest.xsl"/> . . . </xsl:stylesheet> </pre>	<pre> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> . . . <xsl:template match="/" > <xsl:import href="xslttest.xsl"/> <xsl:apply-templates select="//AA" /> </xsl:template> </xsl:stylesheet> </pre>
<pre> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:import href="file1.xsl"/> <xsl:include href="file2.xsl"/> <xsl:template ...> . . . </xsl:stylesheet> </pre>	<pre> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:include href="file2.xsl"/> <xsl:import href="file2.xsl"/> <xsl:template ...> . . . </xsl:stylesheet> </pre>

6.2.10.8 [XSLT-AutoImport] Une feuille XSLT **ne doit pas** s'importer elle-même (directement ou indirectement).

Le comportement d'un import en « boucle » est variable selon les *frameworks* de transformation XSLT. En outre, ce genre de pratique nuit à la compréhension, déjà peu aisée, des feuilles de styles.

Pour toutes ces raisons, une feuille de styles ne doit pas s'importer elle-même.

Contre-Exemple

```
<!-- fichierA.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="fichierA.xsl"/>
  . . .
</xsl:stylesheet>

<!-- fichierB.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="fichierA.xsl"/>
  . . .
</xsl:stylesheet>

<!-- fichierA.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="fichierB.xsl"/>
  . . .
</xsl:stylesheet>
```

6.2.10.9 [XSLT-ImportUneFois] Un fichier xsl **ne doit** être importé qu'une fois avec `xsl:import`.







La flexibilité offerte par le mécanisme d'inclusion offre une meilleure lisibilité des documents et évite d'avoir des documents ou des feuilles de styles trop volumineuse.

Importer une feuille de styles signifie que celle-ci sera chargée à partir de son URI et compilée par le processeur XSLT. Importer plus d'une fois une même feuille de styles revient donc à exécuter plusieurs fois cette opération. Par conséquent, il faut essayer autant que possible de définir une organisation des documents pour limiter le nombre de chargements de documents importés.

6.2.11 Règles sur les modèles XSLT (« `xsl:template` »)

6.2.11.1 Finalité, critères de sélection, sources et contraintes

D'une manière générale, une règle se définit, avec l'élément de haut niveau « `xsl:template` », cette section décrit une suite de règles pour la définition de règles XSLT.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-TemplateMatch]	L'attribut « <code>match</code> » de l'élément « <code>xsl:template</code> » doit être défini, sauf si un nom de règle a été défini avec l'attribut « <code>name</code> ».	• W3C	NOR	
[XSLT-TemplateMatchExpression]	La valeur de l'attribut « <code>match</code> » de l'extension « <code>xsl:template</code> » ne doit pas contenir d'expression de type « <code>VariableReference</code> ».	• W3C	NOR	
[XSLT-TemplateModeEtMatch]	L'attribut « <code>mode</code> » de l'élément « <code>xsl:template</code> » ne doit pas être défini si l'attribut « <code>match</code> » est absent.	• W3C	NOR	
[XSLT-XpathAxe]	Dans les patterns, on ne doit utiliser que les axes « <code>child</code> », « <code>attribute</code> ».	• W3C	NOR	
[XSLT-CaractereSpeciaux]	Au sein d'une expression les caractères spéciaux doivent être remplacés par leur entités.	• W3C	NOR	
[XSLT-ValeurAttribut]	Dans une instruction « <code>match</code> » utilisant une expression de test sur un attribut, il est fortement recommandé d'utiliser le modèle suivant : « <code>match @attrib[.='val']</code> ».	• Pawson ^[4h]	STD-DGFIP	

6.2.11.2 [XSLT-TemplateMatch] L'attribut « `match` » de l'élément « `xsl:template` » **doit** être défini, sauf si un nom de règle a été défini avec l'attribut « `name` ».

L'attribut « `match` » est un motif (*Pattern*) qui identifie le noeud source ou les noeuds pour lesquels la règle s'applique. L'attribut « `match` » est exigé, à moins que l'élément « `xsl:template` » ait un attribut « `name` ».

Exemple	Contre-Exemple
<pre><xsl:template match="pattern"> ... </xsl:template> <xsl:template name="regletest"> ... </xsl:template></pre>	<pre><xsl:template name="regletest" match="pattern"> ... </xsl:template></pre>

6.2.11.3 [XSLT-TemplateMatchExpression] La valeur de l'attribut « match » de l'extension « xsl:template » **ne doit pas** contenir d'expression de type « *VariableReference* ».

L'usage d'une variable référence n'est pas autorisé pour la définition de l'attribut « match ».

Ci-dessous quelques exemples de définitions à éviter :

Contre exemple

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:param name="mavariabale"/>
  . . .
  <xsl:template match="$mavariabale">
```

6.2.11.4 [XSLT-TemplateModeEtMatch] L'attribut « mode » de l'élément « xsl:template » **ne doit pas** être défini si l'attribut « match » est absent.

L'attribut « mode » permet à un élément d'avoir plusieurs modèles, mais il ne peut être utilisé en l'absence de l'attribut « match ».

Exemple	Contre-Exemple
<pre><xsl:template match="/" mode="m"> ... </xsl:template></pre>	<pre><xsl:template mode="m"> ... </xsl:template></pre>

6.2.11.5 [XSLT-XpathAxe] Dans les patterns, on **ne doit** utiliser que les axes « *child* », « *attribute* ».

Autoriser n'importe quelle expression Xpath donnant un ensemble de noeuds peut devenir très complexe et pèserait sur les performances. C'est la raison pour laquelle les expressions Xpath utilisées dans XSLT devraient être limitées aux axes « *child* » et « *attribute* ».

Contre-Exemple

```
<!--
<!-- cette règle ne retourne pas un ensemble de noeuds -->
<xsl:template match="'1'">
  . . .
</xsl:template>
```

6.2.11.6 [XSLT-CaractereSpeciaux] Au sein d'une expression les caractères spéciaux **doivent** être remplacés par leur entités.

L'usage de caractères spéciaux comme les apostrophes, guillemets ou « < » peut générer des erreurs de type expressions invalides.

Exemple	Contre exemple
<code><xsl:if test="position() &lt; 5"></code>	<code><xsl:if test="position() < 5"></code>
<code><xsl:when test='./@ville="Cap d&apos;Agde" '></code>	<code><xsl:when test='./@ville="Cap d'Agde" '></code>

6.2.11.7 [XSLT-ValeurAttribut] Dans une instruction « match » utilisant une expression de test sur un attribut, il est **fortement recommandé** d'utiliser le modèle suivant : « match @attrib[.='val'] ».

De prime abord, il semble plus naturel de tester la valeur de l'attribut ainsi : « @attrib='valeur' ». Néanmoins ce test retourne une valeur booléenne « true » ou « false ». Ce qui, dans ce contexte, signifierait que le « match » se fait sur la valeur d'un noeud ou d'un élément égal à « true » ou « false », et non son contenu.








Si l'on recherche un élément avec un attribut « attrib » ayant la valeur « val », le code dans l'exemple va effectuer la bonne recherche, alors que dans le contre-exemple on va rechercher les noeuds ayant le nom « true » ou « false ».

Exemple	Contre exemple
<pre> <!-- On cherche un attribut nommée 'att' ayant pour valeur 'val' --> <xsl:template match="@att[.='val']"> </pre>	<pre> <!-- On recherche les noeuds ayant le nom « true » ou « false », (résultat de l'expression) --> <xsl:template match="@attrib='val' "> </pre>

6.2.12 Règles sur les paramètres et variables

6.2.12.1 Finalité, critères de sélection, sources et contraintes

Les règles définies ci-dessous ont pour objectif de normaliser les bonnes pratiques d'utilisation des variables dans une feuille de style XSL.

<i>Réf.</i>	<i>Descriptif</i>	<i>Source(s)</i>	<i>Nature</i>	<i>Contrainte</i>
[XSLT-VariableSelectAttrib]	L'attribut « select » ne doit pas être utilisé pour la définition d'une variable si le contenu de l'élément <code><xsl:variable></code> n'est pas vide.	• W3C	NOR	
[XSLT-VariableReference]	Une variable doit être référencée en précédant son nom du caractère « \$ ».	• W3C	NOR	
[XSLT-VariableNom]	Deux variables ne doivent pas avoir le même nom (si leurs portées se chevauchent).	• W3C	NOR	
[XSLT-ParametreSelectAttrib]	L'attribut select ne doit pas être utilisé pour la définition d'un paramètre si le contenu de l'élément <code><xsl:param></code> n'est pas vide.	• W3C	NOR	
[XSLT-ParametreReference]	Un paramètre doit être référencé en précédant son nom du caractère « \$ ».	• W3C	NOR	
[XSLT-ParamDansTemplate]	Les paramètres utilisés dans une règle nommée (ayant un nom défini avec l'attribut « name ») doivent être définis immédiatement après l'élément « <code><xsl:template></code> ».	• W3C	NOR	
[XSLT-ParamVarPosition]	Il est fortement recommandé d'utiliser l'attribut « select » pour assigner la valeur d'un paramètre ou d'une variable utilisée pour la sélection de la position d'un noeud.	• DMOD	STD-DGFIP	
[XSLT-VariableInvariable]	La valeur d'une variable ne doit pas être changée.	• W3C	NOR	

6.2.12.2 [XSLT-VariableSelectAttrib] L'attribut « *select* » **ne doit pas** être utilisé pour la définition d'une variable si le contenu de l'élément `<xsl:variable>` n'est pas vide.

Si l'élément possède un attribut « *select* », la valeur de l'attribut doit être une expression et la valeur de la variable est l'objet qui résulte de l'évaluation de l'expression. Dans ce cas, le contenu de l'élément doit être vide.

Si l'élément « `xsl:variable` » n'a pas d'attribut « *select* » et a un contenu vide, alors la valeur de la variable est une chaîne vide.

Exemple	Contre exemple
<pre><xsl:variable name="organisme" select="Ministere des Finances"/> <xsl:variable name="age" select="22"/> <xsl:variable name="equipesFrançaises" select="//équipes/équipe[@cn='fr']"/> <xsl:variable name="nomNation"> <xsl:call-template name="labelNation"> . . </xsl:call-template> </xsl:variable> <xsl:variable name="vide"/> est équivalent à <xsl:variable name="vide" select="''"/></pre>	<pre><!-- cette déclaration est erroné --> <xsl:variable name="age" select="22"> <xsl:call-template name="calculAge"> . . . </xsl:call-template> </xsl:variable></pre>

6.2.12.3 [XSLT-VariableReference] Une variable **doit** être référencée en précédant son nom du caractère « *\$* ».

L'utilisation d'une variable se fait toujours en la faisant précéder du caractère « *\$* ».

Exemple	Contre exemple
<pre><xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" version= "1.0" > <xsl:output method="text" /> <xsl:variable name="a">12</xsl:variable> <xsl:template match= "/" > <xsl:variable name="b"> 23</xsl:variable> <xsl:value-of select = "concat(\$a,' + ','\$b,' = ',' \$a+\$b)"/> </xsl:template> </xsl:stylesheet></pre>	<pre><xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" version= "1.0" > <xsl:output method="text" /> <xsl:variable name="a">12</xsl:variable> <xsl:template match= "/" > <xsl:variable name="b"> 23</xsl:variable> <xsl:value-of select = "concat(a,' + ','b,' = ',' a+b)"/> </xsl:template> </xsl:stylesheet></pre>

6.2.12.4 [XSLT-VariableNom] Deux variables **ne doivent pas** avoir le même nom (si leurs portées se chevauche).

A chaque variable est associée une portée. On considère qu'une variable est globale si elle est définie par un élément « `xsl:variable` » de premier niveau, fils de « `xsl:stylesheet` », elle est alors visible dans tout le programme. Une variable locale est, quant à elle, définie dans le corps d'une règle et n'est visible que par le noeud et ses descendants.

On ne peut pas définir deux variables ayant le même nom si leur portée se chevauche (« *shadowing* ») à moins que l'une d'elles soit une variable globale ou que les deux soient globales sans la même préséance.

Exemple	Contre exemple
<pre><xsl:variable name="iterateur" select="1"/> <xsl:for-each select="//film"> <xsl:variable name="iterateur" select="titre"/> <!-- corps de la boucle --> </xsl:for-each></pre>	<pre><xsl:variable name="iterateur" select="1"/> <xsl:for-each select="//film"> <!-- corps de la boucle --> <xsl:variable name="iterateur" select="\$ iterateur + 1"/> </xsl:for-each> <p>Il y a <xsl:value-of select="\$iterateur"/> équipes.</p></pre>

Dans la partie contre-exemple, le code est incorrect car la variable « iterateur » au sein de « xsl:for-each » fait partie de la portée de la première définition. Alors que dans l'exemple, la variable « iterateur » peut être complètement redéfinie avec une portée limitée au noeud « xsl:for-each ».

6.2.12.5 [XSLT-ParametreSelectAttrib] L'attribut **select** ne doit pas être utilisé pour la définition d'un paramètre si le contenu de l'élément `<xsl:param>` n'est pas vide.

Si l'élément « xsl:param » possède un attribut « select », la valeur de l'attribut doit être une expression. La valeur de la variable sera donc le résultat de l'évaluation de l'expression. Dans ce cas, le contenu de l'élément doit être vide.

Note : Si l'élément « xsl:param » n'a pas d'attribut « select » et a un contenu vide, alors la valeur du paramètre est une chaîne vide.

Exemple	Contre exemple
<pre><xsl:param name="organisme" select="Ministere des Finances"> <xsl:param name="age" select="22"> <xsl:param name="equipesFrançaises" select="//équipes/équipe[@cn='fr']"> <xsl:param name="nomNation"> <xsl:call-template name="labelNation"> . . </xsl:call-template> </xsl:param> <xsl:param name="x"/> est equivalent à <xsl:param name="x" select="''"/></pre>	<pre><!-- cette déclaration est erroné --> <xsl:param name="age" select="22"> <xsl:call-template name="calculAge"> . . . </xsl:call-template> </xsl:param></pre>

6.2.12.6 [XSLT-ParametreReference] Un paramètre doit être référencé en précédant son nom du caractère « \$ ».

L'utilisation d'un paramètre se fait toujours en le faisant précéder du caractère « \$ ».

Exemple	Contre exemple
<pre><xsl:template name="total"> <xsl:param name="param"> <xsl:element name="strong"> Le Total est: <xsl:value-of select="\$param"/> </xsl:element></pre>	<pre><xsl:template name="total"> <xsl:param name="param"> <xsl:element name="strong"> Le Total est: <xsl:value-of select="param"/> </xsl:element></pre>

6.2.12.7 **[XSLT-ParamDansTemplate]** Les paramètres utilisés dans une règle nommée (ayant un nom défini avec l'attribut « name ») **doivent** être définis immédiatement après l'élément « <xsl:template> ».

Un élément « <xsl:param> » doit être déclaré comme un enfant immédiat d'un élément « <xsl:template> ». S'il n'est pas déclaré comme un enfant immédiat, la valeur de l'élément « <xsl:param> » n'est pas accessible et une erreur se produit.

Il est important de déclarer tout les paramètres qui peuvent être passés à la règle.

Note : La valeur spécifiée dans l'élément « xsl:param » est une valeur par défaut à des fins de liaison. Lorsque une règle contenant « xsl:param » est appelée, des paramètres sont transmis, et sont utilisés à la place des valeurs par défaut. Il est donc possible, et encouragé autant que possible, de fournir des valeurs par défaut aux paramètres.

Exemple	Contre exemple
<pre><xsl:template name="getcount"> <xsl:param name="counted"> <xsl:value-of select="count(//book)"/> </xsl:param> <xsl:element name="strong"> Total Book Count: <xsl:value-of select="\$counted"/> </xsl:element></pre>	<pre><xsl:template name="getcount"> <xsl:element name="strong"> <xsl:param name="counted"> <xsl:value-of select="count(//book)"/> </xsl:param> Total Book Count: <xsl:value-of select="\$counted"/> </xsl:element> </xsl:template></pre>

6.2.12.8 **[XSLT-ParamVarPosition]** Il est **fortement recommandé** d'utiliser l'attribut « select » pour assigner la valeur d'un paramètre ou d'une variable utilisée pour la sélection de la position d'un noeud.

Lorsqu'on utilise des variables pour indiquer des positions, il faut être prudent à la manière dont sera définie le contenu de ces variables. En effet, XSL n'est pas un langage fortement typé, l'interprétation de la position, à partir d'une variable erronée, peut aboutir à des défauts dans l'applicatif très complexes à détecter (et reproduire).

Exemple	Contre exemple
<pre><xsl:variable name="n" select="2"/> ... <xsl:value-of select="item[\$n]"/></pre>	<pre><xsl:variable name="n">2</xsl:variable> ... <xsl:value-of select="item[\$n]"/></pre>
<pre><xsl:param name="n" select="2"/> ... <xsl:value-of select="item[\$n]"/> or <xsl:param name="n">2</xsl:param> ... <xsl:value-of select="item[number(\$n)]"/></pre>	<pre><xsl:param name="n">2</xsl:param> ... <xsl:value-of select="item[\$n]"/></pre>

6.2.12.9 [XSLT-VariableInvariable] La valeur d'une variable **ne doit pas** être changée.



Les variables XSLT sont immuables et lorsque des valeurs leurs sont assignées, ces dernières ne peuvent changer, du moins dans leur domaine de visibilité ou de portée.

Exemple
<pre><xsl:call-template name="compteur"> <xsl:with-param name="iteration" select="0"/> <xsl:with-param name="fin" select="3"/> </xsl:call-template> <xsl:template name="compteur"> <xsl:param name="iteration"/> <xsl:param name="fin"/> <xsl:if test="\$iteration < \$fin"> <xsl:value-of select="'bonjour!'" /> <xsl:call-template name="compteur"> <xsl:with-param name="iteration" select="\$iteration + 1"/> <xsl:with-param name="fin" select="\$fin"/> </xsl:call-template> </xsl:if> </xsl:template></pre>

6.2.13 Règles sur le déclenchement de motifs (template) XSLT

6.2.13.1 Finalité, critères de sélection, sources et contraintes

Les règles suivantes s'intéressent aux mécanismes de déclenchement et d'appel de règles.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-callTemplateName]	L'instruction de déclenchement d'une règle, « xsl:call-template », doit contenir le nom (QName) de la règle invoquée.	• W3C	NOR	
[XSLT-callTemplateParamNonConnu]	Il est fortement recommandé de ne pas invoquer une règle avec des paramètres non connus.	• W3C	NOR	

6.2.13.2 [XSLT-callTemplateName] L'instruction de déclenchement d'une règle, « `xsl:call-template` », **doit** contenir le nom (*QName*) de la règle invoquée.

Exemple	Contre-Exemple
<code><xsl:call-template name="nom"/></code>	<code><xsl:call-template/></code>

6.2.13.3 [XSLT-callTemplateParamNonConnu] Il est **fortement recommandé** de ne pas invoquer une règle avec des paramètres non connus.

Lors du déclenchement d'une règle via « `xsl:template` », la spécification XSLT 1.0 autorise de fournir des paramètres non connus par la règle; ils sont en fait ignorés. Pour des raisons de lisibilité et de cohérence, il est fortement recommandé de ne pas définir de tels paramètres.

La règle suivante

```
<xsl:template name="mafonction">
  <xsl:param name="param1"/>
  <xsl:param name="param2"/>
  . . .
</xsl:template>
```


peut être invoquée comme décrit ci-dessous, mais nous recommandons de respecter le nombre exact de paramètres avec leur nom.

Exemple	Contre-Exemple
<pre><xsl:call-template name="mafonction"> <xsl:with-param name="param1" select="valeur1"/> <xsl:with-param name="param2" select="valeur2"/> </xsl:call-template></pre>	<pre><xsl:call-template name="mafonction"> <xsl:with-param name="param1" select="valeur1"/> <xsl:with-param name="param2" select="valeur2"/> <xsl:with-param name="param3" select="valeur2"/> </xsl:call-template></pre>

6.2.14 Règles sur le format de sortie

6.2.14.1 Finalité, critères de sélection, sources et contraintes

La règle suivante s'intéresse au contenu de la feuille de style afin d'assurer une sortie valide.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-BienForme]	Le contenu complet d'un document XSLT doit être bien formé, comme tout document XML.	• W3C	NOR	

6.2.14.2 [XSLT-BienForme] Le contenu complet d'un document XSLT **doit** être bien formé, comme tout document XML.

Par défaut, les documents générés par un processeur XSLT sont au format XML, et peuvent être définis avec un autre format, en l'occurrence HTML ou « text ». Ainsi lorsque le format de sortie est XML, il est important que l'ensemble du document soit bien formé. Cela inclut aussi bien les éléments XSL que les autres éléments ou littéraux présents dans la feuille de style.

Contre-Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="...">
  <xsl:template match="/">
    <html>
      <head>Entete</head>
      <body>
        <p>
          <h1> </h1>
          <br>
        </body>
      </html>
    </xsl:template>
  </xsl:stylesheet>
```

6.2.15 Règles sur la production de l'arbre de sortie

6.2.15.1 Finalité, critères de sélection, sources et contraintes

L'arbre de sortie généré par un processeur XSLT regroupe plusieurs types d'éléments de type XML. Les règles suivantes expliquent comment optimiser leur production et comment assurer que le contenu de document de sortie est bien valide.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-GenerationNomElement]	Si le nom du noeud à générer est connu, il est recommandé d'utiliser le nom littéral pour le produire directement dans l'arbre de sortie.	• XSLT Tips ^[4f]	STD-DGFIP	★
[XSLT-ElementCalcule]	Il est recommandé d'utiliser «xsl:element» pour la génération d'un élément si celui-ci peut être calculé.	• Tutorial XSLT ^[4g]	STD-DGFIP	★
[XSLT-ElementNomAccolade]	La valeur d'un attribut qui est interprétée comme la valeur d'un élément résultat dans une expression doit être encadrée par des accolades ({}).	• W3C	NOR	★★★

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-ValueOfEscapeDesactive]	On ne doit pas désactiver l'échappement en sortie des caractères spéciaux pour l'élément « xsl:value-of ».	• DMOD	STD-DGFIP	
[XSLT-AttribApresElementsEnfants]	On ne doit pas ajouter un attribut à un élément après que des enfants lui aient été rajoutés.	• W3C	NOR	
[XSLT-AttribEtNoeudElement]	On ne doit pas ajouter un attribut à un noeud qui n'est pas un élément.	• W3C	NOR	
[XSLT-AttribEtNoeudTexte]	On ne doit pas créer des noeuds autres que des noeuds de texte durant l'instanciation du contenu de l'élément « xsl:attribute ».	• W3C	NOR	
[XSLT-TextEscapeDesactive]	On ne doit pas désactiver l'échappement en sortie des caractères spéciaux pour l'élément « xsl:text ».	• DMOD	STD-DGFIP	
[XSLT-Comment]	Le texte mis dans le contenu de l'instruction « xsl-comment » ne doit pas contenir la suite de caractères « -- » ou terminer par le caractère « - ».	• W3C	NOR	
[XSLT-InstTraitNoeudText]	L'instanciation du contenu de « xsl:processing-instruction » ne doit pas créer un noeud autre qu'un noeud textuel.	• W3C	NOR	
[XSLT-InstTraitCorrect]	L'instanciation du contenu de « xsl:processing-instruction » ne doit pas contenir la chaîne « ?> ».	• W3C	NOR	

6.2.15.2 [XSLT-GenerationNomElement] Si le nom du noeud à générer est connu, il est recommandé d'utiliser le nom littéral pour le produire directement dans l'arbre de sortie.

Un élément littéral est automatiquement mis dans l'arbre de sortie, comme décrit dans l'exemple suivant :

Exemple de littéral	Arbre de sortie
<pre><xsl:template match="bar"> <foo> .../... </foo> </xsl:template></pre>	<pre><foo> .../... </foo></pre>

De la même manière, l'instruction « *xsl:element* » peut générer le même résultat :

Contre exemple (utilisation de <i>xsl:element</i>)	Arbre de sortie
<pre><xsl:template match="bar"> <xsl:element name="foo"> .../... </xsl:element> </xsl:template></pre>	<pre><foo> .../... </foo></pre>

Remarque : Il ne s'agit pas de proscrire « *<xsl:element>* », juste son usage abusif.

6.2.15.3 [XSLT-ElementCalcule] Il est recommandé d'utiliser « *xsl:element* » pour la génération d'un élément si celui-ci peut être calculé.

L'élément « *xsl:element* » permet la création d'un élément ayant un nom calculé. Ainsi il est possible d'insérer un élément dans un arbre de sortie sans forcément connaître au préalable sa valeur ou son nom.

L'exemple ci-dessous montre la flexibilité offerte par « *xsl:element* », qui permet de générer le nom d'un élément en fonction de la valeur d'un attribut.

Exemple	Contre-Exemple
<pre><xsl:stylesheet version = '1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'> <xsl:template match="/"> <xsl:for-each select="//text"> <xsl:element name="{@taille}"> <xsl:value-of select="."/> </xsl:element> </xsl:for-each> </xsl:template> </xsl:stylesheet></pre>	<pre><xsl:stylesheet version = '1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'> <xsl:template match="/"> <xsl:for-each select="//text"> <xsl:choose> <xsl:when test="@size='H1'"> <H1> <xsl:value-of select="."/> </H1> </xsl:when> <xsl:when test="@taille='H3'"> <H3> <xsl:value-of select="."/> </H3> </xsl:when> <xsl:when test="@taille='b'"> <xsl:value-of select="."/> </xsl:when> <xsl:when test="@taille='sub'"> <sub> <xsl:value-of select="."/> </sub> </xsl:when> <xsl:when test="@taille='sup'"> <sup> <xsl:value-of select="."/> </sup> </xsl:when> </xsl:choose> </xsl:for-each> </xsl:template></pre>

Les deux modèles décrits ci-dessus produisent le même résultat pour le fichier XML suivant :

Exemple fichier XML	Résultat – Arbre de sortie
<pre><source> <text taille="H1">Header1</text> <text taille="H3">Header3</text> <text taille="b">Bold text</text> <text taille="sub">Subscript</text> <text taille="sup">Superscript</text> </source></pre>	<pre><H1>Header1</H1> <H3>Header3</H3> Bold text <sub>Subscript</sub> <sup>Superscript</sup></pre>

6.2.15.4 [XSLT-ElementNomAccolade] La valeur d'un attribut qui est interprétée comme la valeur d'un élément résultat dans une expression **doit** être encadrée par des accolades ({}).

Un élément XML à générer peut inclure un attribut. La valeur à générer de ce dernier peut être déterminée à partir d'une variable XSLT ou d'une expression XPath à condition de l'encadrer par des accolades ({}). Le modèle de valeur d'attribut est instancié en remplaçant l'expression et les accolades qui la délimitent par le résultat de l'évaluation de l'expression et la conversion de ce résultat en chaîne de caractères.

L'exemple suivant crée un élément résultat « img » à partir de l'élément source « photograph » ; la valeur de l'attribut « src » de l'élément « img » est calculée à partir de la valeur de la variable « image-dir » et de la valeur textuelle de l'enfant « href » de l'élément « photograph » ; la valeur de l'attribut « width » de l'élément « img » est calculée à partir de la valeur de l'attribut « largeur » de l'enfant « taille » de l'élément « photograph » :

Exemple	Contre-Exemple
<pre><xsl:variable name="image-dir"> /images </xsl:variable> <xsl:template match="photograph"> </xsl:template></pre>	<pre><xsl:variable name="image-dir"> /images </xsl:variable> <xsl:template match="photograph"> </xsl:template></pre>

Avec le source :

```
<photograph>
  <href>ministere.jpg</href>
  <taille largeur="300"/>
</photograph>
```

Le résultat serait :

```

```

6.2.15.5 [XSLT-ValueOfEscapeDesactive] On **ne doit pas** désactiver l'échappement en sortie des caractères spéciaux pour l'élément « xsl:value-of ».

Par défaut, la méthode de sortie XML utilise l'échappement pour les caractères <,"',> et & de manière à garantir que la sortie soit bien formée au sens XML. L'élément « xsl:value-of » peut recevoir l'attribut « disable-output-escaping » dont les valeurs autorisées sont « yes » ou « no ». La valeur par défaut est « no » ; si la valeur est mise à « yes », alors les caractères <,"',> et & des noeuds textuels générés par instanciation de l'élément « xsl:value-of » ne seront pas échappés. Utiliser la désactivation de l'échappement peut donc conduire à générer des documents mal formés au sens XML.

Contre-Exemple

```
<xsl:value-of disable-output-escaping="yes" select="string('&lt;')"/>
```

L'instruction ci-dessus signifie à la méthode de sortie de produire le caractère « < ».

6.2.15.6 [XSLT-AttribApresElementsEnfants] On **ne doit pas** ajouter un attribut à un élément après que des enfants lui aient été rajoutés.

L'élément « `xsl:attribute` » permet de rajouter des attributs à des éléments résultats. Pour qu'il soit bien pris en compte, il doit être placé avant tout autre élément au risque d'être ignoré ou de générer une erreur selon le processeur XSLT utilisé.

Exemple	Contre-Exemple
<pre><xsl:stylesheet version = '1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'> <xsl:template match="bar"> <foo> <xsl:attribute name ="param"> valeur </xsl:attribute> <xsl:text>Bonjour</text> </foo> </xsl:template> </xsl:stylesheet></pre>	<pre><xsl:stylesheet version = '1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'> <xsl:template match="bar"> <foo> <xsl:text>Bonjour</text> <xsl:attribute name ="param"> valeur </xsl:attribute> </foo> </xsl:template> </xsl:stylesheet></pre>

L'exemple ci-dessus génère en sortie l'élément

```
<foo param="valeur">
  Bonjour
</foo>
```

6.2.15.7 [XSLT-AttribEtNoeudElement] On **ne doit pas** ajouter un attribut à un noeud qui n'est pas un élément.

L'élément « `xsl:attribute` » ne s'applique qu'à des éléments, qu'ils soient générés directement avec leur nom de manière littérale ou avec l'instruction « `xsl:element` ».

Contre-Exemple

```
<xsl:stylesheet version = '1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

  <xsl:template match="bar">
    <foo>
      <xsl:text>Bonjour
        <xsl:attribute name ="param">
          valeur
        </xsl:attribute>
      </text>
    </foo>
  </xsl:template>
</xsl:stylesheet>
```

6.2.15.8 [XSLT-AttribEtNoeudTexte] On **ne doit pas** créer des noeuds autres que des noeuds de texte durant l'instanciation du contenu de l'élément « `xsl:attribute` ».

Le contenu d'un élément « `xsl:attribute` » doit générer un contenu de type texte. Dans le contre-exemple ci-dessous le contenu d'un élément « `xsl:attribute` » retourne une liste de noeuds (*node set*).

Contre-Exemple

```
<xsl:stylesheet version = '1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

  <xsl:template match="bar">
    <foo>
      <xsl:attribute name ="param">
        <xsl-value-of select="/" />
      </xsl:attribute>
      <xsl:text>Bonjour</text>
    </foo>
  </xsl:template>
</xsl:stylesheet>
```

6.2.15.9 [XSLT-TextEscapeDesactive] On **ne doit pas** désactiver l'échappement en sortie des caractères spéciaux pour l'élément « `xsl:text` ».

Par défaut, la méthode de sortie XML utilise l'échappement pour les caractères `<`, `'`, `>` et `&` de manière à garantir que la sortie soit bien formée au sens XML. L'élément « `xsl:text` » peut recevoir l'attribut « `disable-output-escaping` » dont les valeurs autorisées sont « `yes` » ou « `no` ». La valeur par défaut est « `no` ». Si la valeur est mise à « `yes` », alors les caractères `<`, `'`, `>` et `&` des noeuds textuels générés par instanciation de l'élément « `xsl:text` » ne seront pas échappés. Utiliser la désactivation de l'échappement peut donc conduire à générer des documents mal formés au sens XML.

Contre-Exemple

```
<xsl:template match="/">
  <xsl:text disable-output-escaping="yes">
    &lt;malformé&gt;
  </xsl:text>
</xsl:template>
```

La sortie sera alors l'élément `<malformé>` ne possédant pas de correspondant fermé.

6.2.15.10 [XSLT-Comment] Le texte mis dans le contenu de l'instruction « `xsl-comment` » **ne doit pas** contenir la suite de caractères « `--` » ou terminer par le caractère « `-` ».

L'élément commentaire « `xsl:comment` » sert à créer un noeud commentaire dans l'arbre résultant. Le contenu de l'élément « `xsl:comment` » sert de modèle pour la valeur textuelle du noeud commentaire. Une erreur est produite si le contenu résultant de l'instanciation de « `xsl:comment` » contient la chaîne « `--` » ou si elle finit par « `-` ».

Exemple	Contre-Exemple
<pre><xsl:comment> Ceci est commentaire correct </xsl:comment> Resultat <!-- Ceci est un commentaire correct --></pre>	<pre><xsl:comment> Ceci est une -- infraction à la règle </xsl:comment> <xsl:comment> Autre infraction - </xsl:comment></pre>

6.2.15.11 [XSLT-InstTraitNoeudText] L'instanciation du contenu de « xsl:processing-instruction » **ne doit pas** créer un noeud autre qu'un noeud textuel.

Le contenu d'un élément « xsl:processing » doit générer un contenu de type texte. Dans le contre-exemple ci-dessous le contenu d'un élément « xsl:processing » retourne une liste de noeuds (*node set*).

Exemple
<pre><xsl:template match="/"> <xsl:processing-instruction name="xml-stylesheet"> <xsl:text>type="text/xml" href="style.xml"</xsl:text> </xsl:processing-instruction> </xsl:apply-templates/></pre>
Contre-Exemple
<pre><xsl:template match="/"> <xsl:processing-instruction name="xml-stylesheet"> <foo> <xsl:text>type="text/xml" href="style.xml"</xsl:text> </foo> </xsl:processing-instruction> </xsl:apply-templates/></pre>

6.2.15.12 [XSLT-InstTraitCorrect] L'instanciation du contenu de « xsl:processing-instruction » **ne doit pas** contenir la chaîne « ?> ».




L'instruction « xsl:processing » insère dans le document de sortie la balise fermante (« ?> ») d'une instruction de traitement. Insérer la chaîne « ?> » dans le contenu de l'élément « xsl:processing » reviendrait à générer un document mal formé au sens XML.

Exemple
<pre><xsl:template match="/"> <xsl:processing-instruction name="xml-stylesheet"> <xsl:text>type="text/xml" href="style.xml"</xsl:text> </xsl:processing-instruction> </xsl:apply-templates/></pre>
Contre-Exemple
<pre><xsl:template match="/"> <xsl:processing-instruction name="xml-stylesheet"> <xsl:text>type="text/xml" href="style.xml ?>"</xsl:text> </xsl:processing-instruction> </xsl:apply-templates/></pre>

6.2.16 Règles sur les opérateurs et contrôle de programme XSLT

6.2.16.1 Finalité, critères de sélection, sources et contraintes

Les règles suivantes s'intéressent à la façon d'appliquer des opérateurs et des mécanismes de contrôle au sein de motifs XSLT.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-Union]	L'opérateur « » doit être utilisé pour exprimer une union, au même titre que l'opérateur « union » et non pas un OU logique.	• W3C	NOR	
[XSLT-AndOr]	Les opérateurs « and » et « or » doivent être utilisés avec la bonne syntaxe pour tester la valeur de variables.	• W3C	NOR	
[XSLT-LimiterForEach]	Il est recommandé d'utiliser l'élément « xsl:for-each » lorsque le code dépend de la position du contexte.	• Pawson ^[4h]	STD-DGFIP	

6.2.16.2 **[XSLT-Union]** L'opérateur « | » **doit** être utilisé pour exprimer une union, au même titre que l'opérateur « union » et non pas un OU logique.

Il faut noter que le l'opérateur « | » est utilisé pour exprimer une union et non pas un OU logique, comme dans un langage de programmation. Le OU logique est, quant à lui, exprimé avec « or ».

L'opérateur « | » retourne une union de *nodesets* ou liste de noeuds.

Exemple
<pre> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"> <xsl:outputmethod="xml"..."/> <xsl:templatematch="/aaa"> <xxx> <xsl:value-ofselect="bbb union ccc"/> </xxx> <yyy> <xsl:value-ofselect="bbb ccc"/> </yyy> <zzz> <xsl:value-ofselect="bbb *[1] *[2] *[position() != last()]" /> </zzz> </xsl:template> </xsl:stylesheet> </pre>
<pre> <!-- Appliqué au fichier XML suivant: --> <aaa> <bbb>B1</bbb> <bbb>B2</bbb> <ccc>C1</ccc> <ccc>C2</ccc> </aaa> </pre> <p>Le résultat sera</p> <pre> <xxx>B1 B2 C1 C2</xxx> <yyy>B1 B2 C1 C2</yyy> <zzz>B1 B2 C1</zzz> </pre>

6.2.16.3 [XSLT-AndOr] Les opérateurs « and » et « or » doivent être utilisés avec la bonne syntaxe pour tester la valeur de variables.

Utiliser la syntaxe correcte des opérateurs « and » et « or » lors d'une évaluation d'expression.

Exemple	Contre-Exemple
<code><xsl:when test="\$var='a' or \$var='b'"></code>	<code><xsl:when test="\$var='a'or 'b'"></code>

6.2.16.4 [XSLT-LimiterForEach] Il est recommandé d'utiliser l'élément « xsl:for-each » lorsque le code dépend de la position du contexte.

L'instruction « xsl:for-each » est une structure explicite d'itération dans XSLT, l'autre possibilité étant d'appeler récursivement une règle nommée ou bien « xsl:template ». Avec « xsl:for-each » on peut appliquer un groupe d'instructions sur un groupe de noeuds de manière répétitive, mais son usage modifie la position de contexte courant en se positionnant sur le noeud en cours d'évaluation et donc peut mener à de mauvaises performances.

De manière générale, il est intéressant d'utiliser « xsl:for-each » lorsque la position du contexte est pertinente et plus particulièrement lorsque les fonctions « position() » ou « last() » sont utilisées. Dans le cas contraire, il est plus pertinent d'utiliser « xsl:template ».


Exemple

```
<xsl:for-each select="rss/channel/item">
  <Article num="{position()}">
    <xsl:apply-templates select="title" />
  </Article>
</xsl:for-each>
```

6.2.17 Utilisation des *patterns* XSL

6.2.17.1 Finalité, critères de sélection, sources et contraintes

L'objectif de cette règle est de favoriser l'usage des *patterns* XSLT, afin d'obtenir des feuilles de styles plus optimales.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[XSLT-UtiliserDesignPattern]	Pour améliorer et faciliter la conception de feuille de styles, il est recommandé d'utiliser les « design patterns » XSLT.	<ul style="list-style-type: none"> XSLT – Efficient ^[4e] DMOD 	STD-DGFIP	

6.2.17.2 **[XSLT-UtiliserDesignPattern]** Pour améliorer et faciliter la conception de feuille de styles, il est recommandé d'utiliser les « *design patterns* » XSLT.

La tableau ci dessous regroupe l'ensemble des *patterns* XSLT identifiés lors de la rédaction de cette charte. Sauf contre indication, leur utilisation au sein des projets DGFIP est recommandée.

Pattern	Descriptif	Cas d'utilisation
« Méthode de Muench »	Utilisée pour le « groupage » et la suppression de doublons. Le « groupage » est un problème assez commun dans les feuilles de styles XSLT, qui considère par exemple le réarrangement d'une liste d'éléments. La technique de Muench permet de traiter le groupage ainsi que la suppression de doublons.	Tout document XSLT
« Méthode de Kaysian »	Utilisée pour les intersections et la différence. Cette méthode permet d'exprimer l'intersection ou la différence entre deux <i>node-sets</i> en utilisant uniquement XPath. Au lieu d'utiliser des extensions spécifiques offertes par une librairie tel que EXSLT ou le processeur XSLT (e.g la fonction « <i>intersection</i> »), la méthode Kaysian repose l'opération d'ensemble offerte par XSLT, Union, ainsi que XPath et définit les variables : <pre><xsl:variable name="intersection" select="\$ns1[count(. ns2)=count(ns2)]"/> <xsl:variable name="set-difference" select="\$ns1[count(. ns2)!=count(ns2)]"/></pre>	Tout document XSLT
« Méthode de Piez »	Cette technique permet d'éviter la récursivité pour implémenter des boucles.	Tout document XSLT

Pattern	Descriptif	Cas d'utilisation
« Méthode de Oliver Becker »	<p>XPath est assez puissant pour sélectionner une liste de noeuds en se basant sur des conditions complexes, mais manque d'une gestion efficace de gestion des chaînes de caractères. Souvent, on se retrouve à écrire un code de plusieurs lignes avec « <code>xsl:choose</code> » juste pour spécifier « si cas1 utiliser chaine1, si cas2 utiliser chaine2, ..., si casN utiliser chaineN ».</p> <p>Oliver Becker propose, en remplacement de <code>xsl:choose</code>, un mécanisme d'expression conditionnel basé sur la concaténation de chaînes de caractères; ceci ressemble à la formulation ci-dessous :</p> <pre>concat(substring(chaine1,expression(Condition)), substring(chaine2,expression(not(Condition))))</pre>	Tout document XSLT

Une description de l'ensemble de ces patterns avec des exemple peut être trouvée sur http://www.xml.org/sites/www.xml.org/files/xslt_efficient_programming_techniques.pdf.

6.3 Règles relatives à XSL-FO

Lors de la rédaction de cette charte, peu de bonnes pratiques industrielles, relatives à l'utilisation de XSL-FO, ont été identifiées. En outre, les projets contactés, utilisant XSL-FO, n'ont pas remonté de bonne pratique spécifique au contexte DGFIP, ni même de difficultés rencontrées et adressables par ce présent document.

En conséquence, ce chapitre ne contient, pour le moment, aucune règle sur XSL-FO.

7 Règles relatives à la sécurité

7.1 Finalité, critères de sélection, sources et contraintes

En tant que tels, les documents XML sont rarement exécutés à l'exception notable des fichiers XSLT. Néanmoins, ils présentent, comme tout élément d'un applicatif, un certain nombre de faiblesses en terme de sécurité. Les règles de ce chapitre ont donc pour objectif d'assurer le respect d'un certain nombre de bonnes pratiques visant à interdire le détournement de documents XML ou de scripts XSL.

7.2 Règles déjà citées et ayant un impact positif sur la sécurité

7.2.1 Les règles XML, déjà citées, relatives à la sécurité

7.2.1.1 Structuration du document

Dans ce contexte, le respect des règles [PROLOGUE-Standalone], [CONCEP-DocumentIndépendant] et [CONCEP-PasDeBinaire] prend un nouveau sens et devient d'autant plus essentiel. On notera en particulier que l'externalisation systématique des contenus binaires permet à des mécanismes de sécurité comme SELinux, de bien contrôler leur utilisation. A titre d'exemple, on peut interdire l'exécution d'un fichier binaire s'il est supposé être une image. Par contre, si le binaire de l'image infectée est embarqué dans un document XML, un tel mécanisme est inefficace.

7.2.1.2 Utilisation des entités et instructions de traitements

Les entités étant remplacées automatiquement par la plupart des *frameworks*, elles deviennent, de facto, dans un contexte de sécurité, un point d'entrée pour du code malicieux. Le même raisonnement s'applique d'autant plus aux instructions de traitement. Là encore, il est donc essentiel de s'assurer du respect des règles [XML-InstructionDeTraitementInterdit] et [XML-ENC-EntitesNecessaires].

7.2.1.3 Publication de document XML

En outre, bien que cela ne concerne pas réellement les technologies XML en tant que telles mais la façon dont elles sont utilisées, il est important de respecter les règles :

- [GEN-PUB-ReferenceAccessible] : pour permettre des politiques claires de sécurité où les URLs des dépendances seront clairement définies et les autres simplement interdites ;
- [GEN-PUB-HttpEnc] : pour permettre à des mécanismes de sécurité tel que SELinux d'interdire, par exemple, certaines actions non conformes à ce type de requête.

7.2.2 Les règles XSD, déjà citées, relatives à la sécurité

7.2.2.1 Validation des documents XML

La validation des documents XML, dans un contexte de sécurité, est un point essentiel. Un document XML non validé accepte par définition « tout et n'importe quoi » et forme donc un point d'entrée fantastique pour injecter un code malicieux ou une URL manipulée. Le respect de la règle [XSD-Obligatoire] est donc crucial. En outre, comme le recommande la règle [XSD-PATTERN-

CatchAllElement], il est important de limiter et de structurer les sous parties « dynamiques » d'un document XML.

7.2.2.2 Schéma XSD « statique »

Le schéma lui même peut devenir un point d'attaque s'il autorise des mécanismes de substitution ou de redéfinition. Dans ce contexte, il est important de respecter les règles :

- [XSD-TYPE-PasDeSubstitution] ;
- [XSD-TYPE-PasDeRestrictionDeType] ;
- [XSD-TYPE-PasDeValeursParDefautNiFixe] ;
- [XSD-TYPE-PasDeRedefinitionDeType].

7.2.2.3 Espace de noms des XSD

Enfin, le respect des règles suivantes permet de réduire encore plus les points d'entrée au code malicieux voulant exploiter le schéma XML en lui même :

- [XSD-Namespace-EspaceDeNomParDefaut] ;
- [XSD-NamespaceMêmeEspaceDeNom] ;
- [XSD-Namespace-BlockDefaultAll] ;
- [XSD-Namespace-ElementFormDefaultQualified].




7.2.3 Les règles XPath et XSL, déjà citées, relatives à la sécurité

Comme décrit dans le chapitre sur XSL, l'utilisation de l'instruction « `xsl:import` » permet de redéfinir des traitements pourtant déjà définis. Il est bien évident que ce genre de mécanisme est à proscrire car il permet d'injecter (par exemple, par la manipulation d'une URL), un code malicieux, qui sera exécuté à la place du code normalement importé.

Ainsi, dans ce contexte le respect des règles [XSLT-ImportAvecMode], [XSLT-ImportUneFois] et [XSLT-Import] est crucial.

7.3 Règles spécifiques à la sécurité

Les règles suivantes sont spécifiques à la sécurité et viennent compléter les règles déjà citées.

Réf.	Descriptif	Source(s)	Nature	Contrainte
[SECURITE-InjectionURLs]	Dans un script XSL, les URLs ne doivent pas être construites à partir de données saisies par un utilisateur.	• DMOD	STD-DGFIP	
[SECURITE-InjectionXPath]	Dans un script XSLT, les requêtes XPath ne doivent pas être construites à partir de variables utilisateurs	• DMOD	STD-DGFIP	
[SECURITE-TransformationUtilisateur]	Une transformation XSL ne devrait pas être définie et déclenchée par un utilisateur.	• DMOD	STD-DGFIP	

7.3.1 [SECURITE-InjectionURLs] Dans un script XSL, les URLs **ne doivent pas** être construites à partir de données saisies par un utilisateur.

La principale source de faiblesse vient de la manipulation des URLs. Les technologies XML utilisent beaucoup ces dernières, et il est donc possible, en exploitant cette faiblesse dans les *frameworks* XML utilisés, de manipuler ces dernières pour charger ou utiliser un autre document.

Dans un script XSL, une URL peut être construite à partir de variables dont la valeur est le résultat de la saisie d'un utilisateur. Par conséquent, par des techniques similaires à l'injection SQL, il est possible de manipuler des URLs invalides ou modifiées. Ces dernières formeront alors un point d'entrée à un code malicieux.

7.3.2 [SECURITE-InjectionXPath] Dans un script XSLT, les requêtes XPath **ne doivent pas** être construites à partir de variables utilisateurs

De même qu'avec les URLs, il est possible « d'injecter » du code malicieux dans des requêtes XPath. En effet, comme le rappelle la règle [XPATh-SlashSlash], certains mécanismes de XPath peuvent être extrêmement consommateurs de ressources, ouvrant la porte à des attaques par déni de service.

7.3.3 [SECURITE-TransformationUtilisateur] Une transformation XSL **ne devrait pas** être définie et déclenchée par un utilisateur.

Certaines solutions logicielles définissent, par exemple, des langages de macro permettant de générer une XSLT qui traitera des données soumises par un utilisateur. Ce genre de mécanisme est à proscrire. En effet, il est aisé de détourner les macros pour en faire un code malicieux qui sera par la suite exécuté par le serveur hébergeant l'application, ceci avec les privilèges associés au processus d'exécution du serveur.

8 Annexes

8.1 Bibliographie

Bibliographie				
N°	Version	Date	Référence	Titre
1	1.2.4	24/08/2008	http://publicop.eole.dgi/doc615	Charte d'architecture pour le système d'information fiscal
1b	3.5	10/07/2007	http://publicop.eole.dgi/browse.php?ffFolderId=7562	Matrice OS / Navigateur (compatibilité avec les modules applicatifs Copernic)
1c	1.1	31/05/2005	N/A	Guide UML-XML (ADAE)
2	2.0	21/09/2005	http://publicop.eole.dgi/doc186	Guide de modélisation applicative
2b	0.1	02/06/2006	http://vulcain.dev.impos/	Matrice Technologique Projets
2	1.0	10/09/2007	http://publicop.eole.dgi/doc25974	Charte de développement ECMAScript (JavaScript)
2e	1.0	29/02/2008	http://publicop.eole.dgi/doc28992	Charte de développement HTML/CSS
2f	1.4	16/06/2006	http://publicop.eole.dgi/doc15185	Glossaire Architecture Applicative
2g	B	03/2005	Guide de developpement.zip sur Venezia	Copernic, GAIA : Normes, standards et préconisations de développement
2g	1.4	11/2003	ADOV2_NOR_OO1.1.4.doc sur Venezia	Copernic, ADONIS 2, Normes de développement J2EE
2h	1.0	07/2005	http://publicop.eole.dgi/doc1487	Note : DASP/050451 Normes et standards HTML dans la conception d'applicatif
2i	1.2	07/2007	http://publicop.eole.dgi/doc26153	Note : DASP - synthèse UTF-8
2j	1.0	27/09/2005	http://publicop.eole.dgi/doc4709	Note : DASP - Norme encodage UTF8
2k	N/A	17/04/2003	http://www.unicode.org/reports/tr15/tr15-23.html	Unicode Normalization Forms – Annexe 15 de la spécification Unicode 4.0
2l	N/A	11/09/2005	http://norman.walsh.name/2004/09/30/xml11	XML 1.1 – Dead on arrival

Bibliographie				
2m	N/A	06/05/2008	http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references#Predefined_entities_in_XML	Liste des entités prédéfinies par XML
3	4.01	12/1999	http://www.w3.org/TR/2006/REC-xml-20060816/	La spécification XML 1.0 (4ième édition)
3b	1.0	01/2000	http://www.w3.org/TR/2006/REC-xml11-20060816/	Spécifications XML 1.0 (2ième édition)
3c	N/A	02/05/2001	http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/	Spécifications XML Schema
3d	1.0	11/1999	http://www.w3.org/TR/xpath	La spécification XPath 1.0
3e	1.0	11/1999	http://www.w3.org/TR/xslt	La spécification XSLT 1.0
3f	2.0	01/2007	http://www.w3.org/TR/xpath20/	La spécification XPath 2.0
3g	2.0	01/2007	http://www.w3.org/TR/xslt20/	La spécification XSLT 2.0
4	N/A	09/2003	http://search.barnesandnoble.com/Effective-XML/Elliotte-Rusty-Harold/e/9780321150400	« Effective XML » par <i>Elliotte Rusty Harold</i>
4a	1.3	25/10/2004	http://www.medbiq.org/technology/tech_architecture/xmldesignguidelines.pdf	« XML Schema Design Guidelines », Consortium MedBiquitous
4b	N/A	09/2007	http://www.eyrolles.com/Informatique/Livre/9782744072369/livre-xml.php	« XML – Synthèse de cours », de G.Chagnon et F.Nolot
4c	N/A	08/2000	http://www.dunod.com/pages/ouvrages/ficheouvrage.asp?id=46104	« Formation à ... XML » de M.J. Young
4d	N/A	08/2002	http://www.xml.com/lpt/a/1018	« Top Ten Tips to Using XPath and Xpointer » de John E. Simpson
4e	N/A	N/A	http://www.xml.org/sites/www.xml.org/files/xslt_efficient_programming_techniques.pdf	« XSLT – Efficient Programming Techniques » de Prathit Bondre

Bibliographie				
4f	N/A	09/2007	http://www.onenaught.com/posts/23/xslt-tips-for-cleaner-code-and-better-performance	« XSLT Tips for Cleaner Code and Better Performance » de Anup Shah
4g	N/A	12/2000	http://www.zvon.org/xml/XSLTutorial/Output_fre/contents.html	« Tutorial XSLT » de Miroslav Nic
4h	N/A	N/A	http://www.dpawson.co.uk/xsl/sect2/sect21.html	« XSLT Questions and Answers - FAQ » de Dave Pawson
4i	N/A	02/2002	Edition O'Reilly	« Comprendre XSLT » de Bernd Amman et Philippe Rigaux

8.2 Lexique

Lexique	
Terme	Définition
charte	Une charte est l'ensemble de règles et principes fondamentaux d'une institution officielle (Définition Wikipedia).
encodage	Un encodage associe les symboles utilisés par une langue naturelle (comme un alphabet ou un syllabaire) avec un ensemble de code binaire.
XML	eXtended Markup Language ; en français, langage de balisage étendu
<i>parser</i>	Un <i>parser</i> ou analyseur syntaxique est un programme qui analyse la structure d'un fichier XML pour exposer les informations contenues. De même, c'est souvent ce même analyseur syntaxique qui sera utilisé pour produire le document XML.
<i>framework</i>	En informatique, un <i>framework</i> est un espace de travail modulaire. C'est un ensemble de bibliothèques, d'outils et de conventions permettant le développement d'applications. Il fournit suffisamment de briques logicielles et impose suffisamment de rigueur pour pouvoir produire une application aboutie et facile à maintenir. Ces composants sont organisés pour être utilisés en interaction les uns avec les autres. (http://fr.wikipedia.org/wiki/Framework)
entité	Les entités externes, issues de SGML, permettent d'insérer des éléments externes au document par le biais d'un identifieur. Elles sont résolues à priori par un <i>parser</i> validant, avant tout traitement du document XML.
prologue	En XML, le prologue est constitué de la déclaration <code><?xml version="1.0"?></code> , et de la déclaration de type de document (DOCTYPE). (http://fr.wikipedia.org/wiki/XML#Le_prologue)
<i>binding</i>	Un <i>binding</i> (qui est un terme anglais désignant l'action de lier des éléments entre eux) signifie en informatique le fait de permettre l'utilisation d'une bibliothèque logicielle dans un autre langage de programmation que celui avec lequel elle a été écrite. (http://fr.wikipedia.org/wiki/Binding)
<i>PSVI</i>	Le « <i>Post Schema Validation</i> » correspond à une recommandation du W3C qui encourage les parsers XML à fournir, à l'issue de la validation d'un flux XML, l'ensemble des informations de typage sur les données validées.
<i>unicode</i>	Unicode est une norme informatique, développée par le Consortium Unicode, qui vise à donner à tout caractère de n'importe quel système d'écriture de langue un nom et un identifiant numérique, et ce de manière unifiée, quelle que soit la plate-forme informatique ou le logiciel.
espace de noms	Un espace de noms (namespace) est un ensemble de ce qui est désignable dans un contexte donné par une méthode d'accès donnée faisant usage de noms symboliques (par exemple des chaînes de caractères avec ou sans restriction d'écriture). (http://fr.wikipedia.org/wiki/Espace_de_noms)

Lexique	
encapsulation de liste	Dans le contexte de cette charte, on entend par encapsulation de liste, le fait d'utiliser une balise spécifique pour encadrer une série contigue de balises identiques.
document	Un document XML désigne un ensemble de données XML, quelque soit son état de représentation physique (fichier, chaîne de caractères, flux,...)
<i>IDE</i>	En anglais, <i>Integrated Development Environment</i> : environnement de développement intégré. Exemples : Eclipse, NetBeans.

8.3 Sommaire détaillé

1	<u>Présentation de la charte de développement XML/XSL</u>	7
1.1	<u>Contexte et objectifs stratégiques sous-jacents</u>	7
1.2	<u>Le périmètre de la charte</u>	7
1.3	<u>Les versions XML/XSL supportées</u>	8
1.3.1	<u>Les différentes normes ou recommandations XML</u>	8
1.3.2	<u>Les différentes normes ou recommandations XSL</u>	8
1.3.3	<u>Les usages</u>	8
1.4	<u>Les métiers ciblés</u>	10
1.5	<u>Application de la charte</u>	10
1.6	<u>Processus d'évaluation ou d'audit</u>	10
1.7	<u>Gestions des amendements et des évolutions</u>	10
2	<u>Démarche et conventions adoptées</u>	11
2.1	<u>Sources</u>	11
2.2	<u>Structure</u>	11
2.3	<u>Critères de sélection des règles</u>	12
2.4	<u>Conventions</u>	13
2.4.1	<u>Nature des règles</u>	13
2.4.2	<u>Contraintes d'utilisation</u>	13
3	<u>Règles sur le langage de balisage XML</u>	15
3.1	<u>Introduction</u>	15
3.2	<u>Prologue du document</u>	15
3.2.1	<u>Finalité, critères de sélection, sources et contraintes</u>	15
3.2.2	<u>[PROLOGUE-XmlValid] Le document doit être conforme au format XML 1.0</u>	16
3.2.3	<u>[PROLOGUE-Entete] Le document XML doit contenir un entête</u>	16
3.2.4	<u>[PROLOGUE-XmlVersion] Le document doit utiliser la version 1.0 de la spécification XML, de manière explicite</u>	16
3.2.5	<u>[PROLOGUE-encodage] Le document doit utiliser l'encodage UTF-8 de manière explicite</u>	16
3.2.6	<u>[PROLOGUE-Standalone] L'utilisation de la déclaration standalone="yes" est recommandée dans le document</u>	16
3.2.7	<u>[XML-EnteteFichier] Un document XML doit contenir un entête de fichier avec les informations minimales sur son propos</u>	17
3.2.8	<u>Exemple récapitulatif</u>	17
3.3	<u>L'encodage du document</u>	17
3.3.1	<u>[XML-ENC-FormeNormaleC] Un document XML doit utiliser la Forme Normale C (« Normalization Form C »)</u>	18
3.3.3	<u>[XML-GuillemetEtApostrophe] On ne doit utiliser les entités &apos; et &quot; pour représenter les apostrophes et les guillemets que si nécessaire</u>	19
3.4	<u>Instruction de traitement et entités</u>	20
3.4.1	<u>Finalités, critères de sélection, sources et contraintes</u>	20

3.4.2	[XML-InstructionDeTraitementInterdit] On ne doit pas utiliser les instructions de traitement au sein d'un document XML.	21
3.4.3	[Entite-NePasUtiliserEntites] Pour conserver le document XML autonome, il est recommandé de ne pas utiliser d'entités (à l'exception des entités « & » , « &lt; » , « &gt; » , « &apos; » et « &quot; »).	21
3.5	Utilisation des attributs spéciaux (xsi: et xml:)	21
3.5.1	Finalités, critères de sélection, sources et contraintes	21
3.5.2	[XML-NulliteElement] Pour indiquer qu'un élément XML est nul, il est recommandé de ne pas utiliser l'attribut spécial « xsi:nil ».	22
3.5.3	[XML-Espace] Pour préserver les espaces au sein d'un élément, on doit utiliser l'attribut « xml:space ».	22
3.6	Nommage	23
3.6.1	Finalités, critères de sélection, sources et contraintes	23
3.7	Documentation	26
3.7.1	Finalités, critères de sélection, sources et contraintes	26
3.8	Mise en forme	27
3.8.1	Finalités, critères de sélection, sources et contraintes	27
3.8.2	Règles générales sur la mise en forme du document	28
3.8.3	Règles sur les sections CDATA et PCDATA	29
3.8.4	Règles sur le « pretty-printing »	29
3.8.6	[FORM-PRETTYPRINT-CompatibilitéNonXml] Il est recommandé de conserver une présentation qui permette l'analyse, ligne à ligne, du fichier.	32
3.9	Conception de document XML	33
3.9.2	Métriques de conception	34
3.9.2.1	[CONCEP-MaxAttributs] Un élément doit avoir moins de 6 attributs.	34
3.9.3	Relation attribut-élément	34
3.9.3.1	[CONCEP-DonneeUnique] Un attribut doit représenter une donnée unique et atomique, sinon la donnée doit être représentée par un élément.	34
3.9.3.1.1	Est-ce que la donnée est unique ?	34
3.9.3.1.2	Est-ce que la donnée est atomique ?	35
3.9.3.1.3	[CONCEP-NomElementPere] Un attribut ne doit pas avoir le même nom que son élément.	35
3.9.4	Hiérarchie et arborescence	35
3.9.4.1	[CONCEP-ProfondeurArborescence] Une arborescence XML ne doit pas être plus profonde que 10 niveaux.	35
3.9.4.2	[CONCEP-ElementListe] Les éléments « liste » doivent être utilisés pour encadrer un ensemble d'éléments identiques successifs.	36
3.9.5	[CONCEP-ConteneurListe] Le nom des éléments « liste », contenant une série d'éléments identiques, doit toujours être au pluriel.	36
3.9.6	Données modélisées	37
3.9.6.1	[CONCEP-ValeurAtomique] Les données d'un champ XML doivent être aussi atomiques que l'exige la modélisation.	37
3.9.6.2	[CONCEP-PasDeBinaire] Les données binaires d'un document XML doivent être externalisées.	38
3.9.6.3	[CONCEP-UtiliseXHTMLPourLeContenuNarratif] Il est recommandé d'utiliser XHTML pour structurer les données de type texte dans un document.	39
3.9.6.4	[CONCEP-QuandUtiliserCDATA] Il est recommandé de n'utiliser les sections CDATA dans un document XML que si nécessaire.	41
3.9.7	Intégrité du document	42
3.10	Utilisation des patterns et anti-patterns XML	42
3.10.1	Finalité, critères de sélection, sources et contraintes	42
3.10.2	Patterns associés aux règles déjà citées de la charte	42

3.10.3	Patterns en contradiction avec les règles déjà citées de la charte	42
3.10.3.1	« Patterns » recommandant l'utilisation des DTDs	42
3.10.3.2	Autres patterns en contradiction	43
3.11	Performance	47
3.11.1	[PERF-CompresserDocument] Si un document XML volumineux pose des problèmes d'échange sur le réseau, il doit être compressé	47
3.11.2	[PERF-AnalyseEvenementielDesDocumentsVolumineux] Si un document XML est volumineux (plusieurs méga-octets), on doit l'analyser avec SAX	48
3.12	Génération et publication de flux XML	48
3.12.1	[GEN-PUB-FrameworkXml] La génération de document XML doit passer par un framework	49
3.12.3	[GEN-PUB-HttpEnc] L'encodage ne doit pas être précisé dans la réponse HTTP	50
4	Règles d'utilisation des espaces de noms XML	51
4.1	A propos de XML Namespace	51
4.2	Conventions de nommage	51
5	Règles associées à la validation XSD	52
5.1	Finalité, critères de sélection, sources et contraintes	52
5.2	Documents XML et XSD	53
5.2.1	[XSD-Obligatoire] Un document XML doit être associé à une XSD	53
5.2.2	[XSD-PasDeDTD] Un document XML doit utiliser une XSD et non une DTD pour valider un document XML	53
5.3	Un document XSD est avant tout un document XML	53
5.3.1	Prologue, encodage et nommage	53
5.3.2	[XSD-EnteteFichier] Un document XSD doit contenir un entête de fichier avec les informations minimales sur son propos, son rôle et sa maintenance	54
5.3.3	[XSD-CamelCase] Les noms des éléments ou des attributs doivent suivre la convention « camelCase », en commençant par une minuscule et en séparant les mots à l'aide de majuscules	54
5.3.4	Règles de conception XML applicable aux XSD	55
5.4	Structuration du document	55
5.4.1	[XSD-Pattern-VenetianBlind] La structuration d'un fichier XSD doit suivre le pattern « Venetian Blind »	55
5.4.2	[XSD-Pattern-PasBologno] La structuration d'un fichier XSD ne doit pas suivre le pattern « Bologno »	57
5.5	Règles relatives aux Namespaces au sein des XSD	57
5.5.1	[XSD-Namespace-Prefix] Un schéma XSD doit utiliser le préfixe « xsd »	57
5.5.2	[XSD-Namespace-EspaceDeNomParDefaut] Un schéma XSD doit spécifier un espace de noms par défaut	58
5.5.3	[XSD-NamespaceMêmeEspaceDeNom] L'espace de noms d'un schéma XSD doit être le même que son espace de noms par défaut	58
5.5.4	[XSD-Namespace-VersionSchema] Un schéma XSD doit spécifier la version à la fin de l'URI de son espace de noms par défaut	58
5.5.5	[XSD-Namespace-ElementFormDefaultQualified] Un schéma XSD doit définir l'attribut « elementFormDefault » à « qualified »	59
5.5.6	[XSD-Namespace-AttributeFormDefaultUnqualified] Un schéma XSD doit définir l'attribut attributeFormDefault à unqualified	59
5.5.7	[XSD-Namespace-BlockDefaultAll] Un schéma XSD doit définir l'attribut « blockDefault » à « #all »	60
5.6	Règles de définition de type	61


5.6.1	[XSD-Type-TypeDansNom] Un schéma XSD ne doit pas définir les éléments en intégrant leur type à leur nom	61
5.6.2	[XSD-Type-TypeSimple] Un schéma XSD doit utiliser les types simples	62
5.6.3	[XSD-TYPE-PasDeRestrictionDeType] Un schéma XSD ne doit utiliser la restriction que pour les types simples prédéfinis par XSD	63
5.6.4	[XSD-TYPE-PasDeValeursParDefautNiFixe] Il est recommandé qu'un schéma XML n'utilise pas les valeurs par défaut et les valeurs fixes	63
5.6.5	[XSD-TYPE-PasDeRedefinitionDeType] Le schéma XML ne doit pas utiliser de redéfinition de type	63
5.6.6	[XSD-TYPE-PasDeSubstitution] Un schéma XML ne doit pas utiliser les groupes de substitution	64
5.6.7	[XSD-TYPE-NomTypeElementDifferent] Au sein d'une XSD, on ne doit pas utiliser le même nom pour désigner un type et un élément	64
5.7	Règles issues des « patterns » XML	65
5.8	Règles de gestion des commentaires	66
5.8.1	[XSD-COM-UtiliserLesCommentairesXML] Il est recommandé d'utiliser les commentaires XML pour documenter un schéma	66
5.8.2	[XSD-COM-DocumentsAcronymes] Si des acronymes sont utilisés pour les noms des types, ils doivent être documentés	67
5.9	Règles sur l'utilisation des PSVI	67
6	XSL	68
6.1	Règles relatives à XPath	68
6.1.1	Introduction	68
6.1.2	Finalité, critères de sélection, sources et contraintes	69
6.1.2.1	[XPATh-SlashSlash] L'utilisation de « // » n'est pas recommandée dans les expressions XPath	69
6.1.2.2	[XPATh-EnfantImmediat] On ne doit pas utiliser une expression XPath qui commence avec « // » « / » ou « . » si des enfants immédiats d'un noeud sont recherchés	70
6.1.2.3	[XPATh-FiltreAttribut] Un filtrage par attribut ne doit intervenir que dans la dernière étape d'une expression Xpath	71
6.2	Règles relatives à XSLT	71
6.2.1	Introduction	71
6.2.2	Règles XML applicables à un document XSLT	72
6.2.2.1	Prologue, encodage et nommage	72
6.2.3	Règles de conception XML applicables à un document XSLT	72
6.2.4	Règles sur la présentation d'un document XSLT	73
6.2.4.1	Finalité, critères de sélection, sources et contraintes	73
6.2.5	[XSLT-CamelCase] Les noms des éléments ou des attributs doivent être écrits en minuscules, en séparant les mots à l'aide de traits d'union	73
6.2.6	[XSLT-NomSuffix] Un document XSLT doit avoir le suffixe « .xsl »	73
6.2.6.1	[XSLT-Nom] Un document XSLT doit avoir un nom qui rend compte de son rôle	73
6.2.6.2	[XSLT-NomCorrect] Un document XSLT ne doit pas comporter de caractères accentués	74
6.2.7	Règles de documentation	74
6.2.7.1	Finalité, critères de sélection, sources et contraintes	74
6.2.7.2	[XSLT-EnteteFichier] Un document XSLT doit contenir un entête de fichier avec informations minimales sur son propos, son rôle et son fonctionnement	74
6.2.7.3	[XSLT-EnteteRegle] Une règle XSLT doit être commentée afin de comprendre son fonctionnement	75
6.2.7.4	[XSLT-VersionDocument] Un document XSLT doit avoir une variable « version », placée comme premier fils de « stylesheet », qui contient le numéro de version du document	75
6.2.8	Règles sur la Structure d'un document XSLT	76

6.2.8.1	Finalité, critères de sélection, sources et contraintes	76
6.2.8.2	[XSLT-Racine] Un document XSLT doit débuter par l'élément « stylesheet »	76
6.2.8.3	[XSLT-Version] L'élément XSLT racine doit contenir la version XSLT 1.0	77
6.2.8.4	[XSLT-NameSpace-Prefix] Un document XSLT doit définir le préfixe « xsl: »	77
6.2.8.5	[XSLT-NomsElements] Les noms des éléments XSLT, des attributs et des fonctions doivent être en lettres minuscules, et utiliser le trait d'union pour séparer les mots	77
6.2.9	Règles sur les éléments de premier niveau	78
6.2.9.1	Finalité, critères de sélection, sources et contraintes	78
6.2.9.2	[XSLT-PremierNiveau] Les éléments de haut niveau doivent être du type standard et conformes à l'espace de noms XSLT	78
6.2.9.3	[XSLT-TextPremierNiveauInterdit] Il ne doit pas y avoir des noeuds de type texte placés immédiatement sous l'élément racine	79
6.2.9.4	[XSLT-PréfixeProcesseur] Les extensions spécifiques au processeur XSLT ne doivent pas être utilisées	79
6.2.10	Modularité et inclusion de feuille de styles	80
6.2.10.1	Finalité, critères de sélection, sources et contraintes	80
6.2.10.2	[XSLT-UtiliserInclude] L'inclusion de feuilles de styles doit être réalisée à l'aide de l'instruction « xsl:include »	81
6.2.10.3	[XSLT-ImportAvecMode] Il est recommandé de redéfinir les règles, si besoin est, à l'aide de l'attribut « mode »	81
6.2.10.4	[XSLT-Include] L'élément « xsl:include » ne doit être utilisé que comme élément de premier niveau (fils de l'élément « xsl:stylesheet »)	82
6.2.10.5	[XSLT-AutoInclude] Une feuille XSLT ne doit pas s'inclure elle-même (directement ou indirectement)	83
6.2.10.6	[XSLT-IncludeUneFois] Un fichier xsl ne doit être inclus qu'une seule fois	84
6.2.10.7	[XSLT-Import] On ne doit utiliser l'élément « xsl:import » que comme élément de premier niveau, et il doit précéder tous les autres éléments fils de « xsl:stylesheet »	84
6.2.10.8	[XSLT-AutoImport] Une feuille XSLT ne doit pas s'importer elle-même (directement ou indirectement)	85
6.2.10.9	[XSLT-ImportUneFois] Un fichier xsl ne doit être importé qu'une fois avec xsl:import	85
6.2.11	Règles sur les modèles XSLT (« xsl:template »)	86
6.2.11.1	Finalité, critères de sélection, sources et contraintes	86
6.2.11.2	[XSLT-TemplateMatch] L'attribut « match » de l'élément « xsl:template » doit être défini, sauf si un nom de règle a été défini avec l'attribut « name »	86
6.2.11.3	[XSLT-TemplateMatchExpression] La valeur de l'attribut « match » de l'extension « xsl:template » ne doit pas contenir d'expression de type « VariableReference »	87
6.2.11.4	[XSLT-TemplateModeEtMatch] L'attribut « mode » de l'élément « xsl:template » ne doit pas être défini si l'attribut « match » est absent	87
6.2.11.5	[XSLT-XpathAxe] Dans les patterns, on ne doit utiliser que les axes « child », « attribute »	87
6.2.11.6	[XSLT-CaractereSpeciaux] Au sein d'une expression les caractères spéciaux doivent être remplacés par leur entités	87
6.2.11.7	[XSLT-ValeurAttribut] Dans une instruction « match » utilisant une expression de test sur un attribut, il est fortement recommandé d'utiliser le modèle suivant : « match @attrib[.='val'] »	88
6.2.12	Règles sur les paramètres et variables	89
6.2.12.1	Finalité, critères de sélection, sources et contraintes	89
6.2.12.2	[XSLT-VariableSelectAttrib] L'attribut « select » ne doit pas être utilisé pour la définition d'une variable si le contenu de l'élément <xsl:variable> n'est pas vide	90
6.2.12.3	[XSLT-VariableReference] Une variable doit être référencée en précédant son nom du caractère « \$ »	90
6.2.12.4	[XSLT-VariableNom] Deux variables ne doivent pas avoir le même nom (si leurs portées se chevauche)	90

6.2.12.5 [XSLT-ParametreSelectAttrib] L'attribut select ne doit pas être utilisé pour la définition d'un paramètre si le contenu de l'élément <xsl:param> n'est pas vide.....	91
6.2.12.6 [XSLT-ParametreReference] Un paramètre doit être référencé en précédant son nom du caractère « \$ ».....	91
6.2.12.7 [XSLT-ParamDansTemplate] Les paramètres utilisés dans une règle nommée (ayant un nom défini avec l'attribut « name ») doivent être définis immédiatement après l'élément « <xsl:template> ».....	92
6.2.12.8 [XSLT-ParamVarPosition] Il est fortement recommandé d'utiliser l'attribut « select » pour assigner la valeur d'un paramètre ou d'une variable utilisée pour la sélection de la position d'un noeud.....	92
6.2.12.9 [XSLT-VariableInvariable] La valeur d'une variable ne doit pas être changée.....	93
6.2.13 Règles sur le déclenchement de motifs (template) XSLT.....	93
6.2.13.1 Finalité, critères de sélection, sources et contraintes.....	93
6.2.13.2 [XSLT-callTemplateName] L'instruction de déclenchement d'une règle, « xsl:call-template », doit contenir le nom (QName) de la règle invoquée.....	94
6.2.13.3 [XSLT-callTemplateParamNonConnu] Il est fortement recommandé de ne pas invoquer une règle avec des paramètres non connus.....	94
6.2.14 Règles sur le format de sortie.....	94
6.2.14.1 Finalité, critères de sélection, sources et contraintes.....	94
6.2.14.2 [XSLT-BienForme] Le contenu complet d'un document XSLT doit être bien formé, comme tout document XML.....	95
6.2.15 Règles sur la production de l'arbre de sortie.....	95
6.2.15.1 Finalité, critères de sélection, sources et contraintes.....	95
6.2.15.2 [XSLT-GenerationNomElement] Si le nom du noeud à générer est connu, il est recommandé d'utiliser le nom littéral pour le produire directement dans l'arbre de sortie.....	96
6.2.15.3 [XSLT-ElementCalcule] Il est recommandé d'utiliser « xsl:element » pour la génération d'un élément si celui-ci peut être calculé.....	97
6.2.15.4 [XSLT-ElementNomAccolade] La valeur d'un attribut qui est interprétée comme la valeur d'un élément résultat dans une expression doit être encadrée par des accolades ({})......	98
6.2.15.5 [XSLT-ValueOfEscapeDesactive] On ne doit pas désactiver l'échappement en sortie des caractères spéciaux pour l'élément « xsl:value-of ».....	98
6.2.15.6 [XSLT-AttribApresElementsEnfants] On ne doit pas ajouter un attribut à un élément après que des enfants lui aient été rajoutés.....	99
6.2.15.7 [XSLT-AttribEtNoeudElement] On ne doit pas ajouter un attribut à un noeud qui n'est pas un élément.....	99
6.2.15.8 [XSLT-AttribEtNoeudTexte] On ne doit pas créer des noeuds autres que des noeuds de texte durant l'instanciation du contenu de l'élément « xsl:attribute ».....	100
6.2.15.9 [XSLT-TextEscapeDesactive] On ne doit pas désactiver l'échappement en sortie des caractères spéciaux pour l'élément « xsl:text ».....	100
6.2.15.10 [XSLT-Comment] Le texte mis dans le contenu de l'instruction « xsl:comment » ne doit pas contenir la suite de caractères « -- » ou terminer par le caractère « - ».....	100
6.2.15.11 [XSLT-InstTraitNoeudText] L'instanciation du contenu de « xsl:processing-instruction » ne doit pas créer un noeud autre qu'un noeud textuel.....	101
6.2.15.12 [XSLT-InstTraitCorrect] L'instanciation du contenu de « xsl:processing-instruction » ne doit pas contenir la chaîne « ?> ».....	101
6.2.16 Règles sur les opérateurs et contrôle de programme XSLT.....	102
6.2.16.1 Finalité, critères de sélection, sources et contraintes.....	102
6.2.16.2 [XSLT-Union] L'opérateur « » doit être utilisé pour exprimer une union, au même titre que l'opérateur « union » et non pas un OU logique.....	102
6.2.16.3 [XSLT-AndOr] Les opérateurs « and » et « or » doivent être utilisés avec la bonne syntaxe pour tester la valeur de variables.....	103
6.2.16.4 [XSLT-LimiterForEach] Il est recommandé d'utiliser l'élément « xsl:for-each » lorsque le code dépend de la position du contexte.....	103
6.2.17 Utilisation des patterns XSL.....	104
6.2.17.1 Finalité, critères de sélection, sources et contraintes.....	104

6.2.17.2 [XSLT-UtiliserDesignPattern] Pour améliorer et faciliter la conception de feuille de styles, il est recommandé d'utiliser les « design patterns » XSLT.	104
6.3 Règles relatives à XSL-FO.	105
7 Règles relatives à la sécurité.	106
7.1 Finalité, critères de sélection, sources et contraintes.	106
7.2 Règles déjà citées et ayant un impact positif sur la sécurité.	106
7.2.1 Les règles XML, déjà citées, relatives à la sécurité.	106
7.2.1.1 Structuration du document.	106
7.2.1.2 Utilisation des entités et instructions de traitements.	106
7.2.1.3 Publication de document XML.	106
7.2.2 Les règles XSD, déjà citées, relatives à la sécurité.	106
7.2.2.1 Validation des documents XML.	106
7.2.2.2 Schéma XSD « statique ».	107
7.2.2.3 Espace de noms des XSD.	107
7.2.3 Les règles XPath et XSL, déjà citées, relatives à la sécurité.	107
7.3 Règles spécifiques à la sécurité.	108
7.3.1 [SECURITE-InjectionURLs] Dans un script XSL, les URLs ne doivent pas être construites à partir de données saisies par un utilisateur.	108
7.3.2 [SECURITE-InjectionXPath] Dans un script XSLT, les requêtes XPath ne doivent pas être construites à partir de variables utilisateurs.	108
7.3.3 [SECURITE-TransformationUtilisateur] Une transformation XSL ne devrait pas être définie et déclenchée par un utilisateur.	108
8 Annexes.	109
8.1 Bibliographie	109
8.2 Lexique	112
8.3 Sommaire détaillé	114
8.4 Licence du présent document	122

8.4 Licence du présent document




CC creative commons
COMMONS DEED

Paternité - Pas d'Utilisation Commerciale - Partage des Conditions Initiales à l'Identique 2.0 France


Vous êtes libres :

- ♦ de reproduire, distribuer et communiquer cette création au public
- ♦ de modifier cette création


Selon les conditions suivantes :



Paternité. Vous devez citer le nom de l'auteur original.



Pas d'Utilisation Commerciale. Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.



Partage des Conditions Initiales à l'Identique. Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

- ♦ A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.
- ♦ Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.

Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur : copies réservées à l'usage privé du copiste, courtes citations, parodie...)

Ceci est le Résumé Explicatif du [Code Juridique \(la version intégrale du contrat\)](#).

[Avertissement](#) 