

# Standards de développement : Gestion du code source

Date d'application : 12/11/2019

Version : V1.1

---

**Retrouvez-nous sur :**  
[justice.gouv.fr](http://justice.gouv.fr)

## Circuit de validation

<i>Date application</i>	<i>Version</i>	<i>Objet</i>	<i>Rédaction</i>	<i>Vérification</i>	<i>Approbation</i>
12/11/2019	V1.1	Standards de développement	12/11/2019 Jean-Christophe TOFFALONI	12/11/2019 Dominique PAILHAREY	12/11/2019 Stéphanie MAUREY

## Diffusion

<i>Pour action</i>	
<i>Pour information</i>	

## Historique des modifications

<i>Date application</i>	<i>Version</i>	<i>Objet</i>	<i>Rédaction</i>	<i>Vérification</i>	<i>Approbation</i>
12/11/2019	V1.1	<ul style="list-style-type: none"><li>- Ajout « gestion du code source » dans le titre du document pour créer le chapitre des normes spécifiques.</li><li>- Précisions pour éviter les clouds publiques et GitHub (§4.5.3).</li><li>- Précisions sur le regroupement dans un seul référentiel des bibliothèques ayant le même nom complet (§4.2).</li><li>- Normes de config Graddle avec les mêmes contraintes que Maven (ex : profile maven presta/MJ, dev/re7/prod) (§4.5.2).</li><li>- Créer des tags aussi en phase de dev pour identifier des lots ou des étapes longues dans le temps (GitFlow) (§4.5.2).</li><li>- Principe du « sources.list » pour l'automatisation des matrices de compatibilité (§4.5.2).</li></ul>	12/11/2019 Jean-Christophe TOFFALONI	12/11/2019 Dominique PAILHAREY	12/11/2019 Stéphanie MAUREY

## Sommaire

1	Objet .....	3
2	Domaine d'application .....	3
3	Références, définitions et terminologie .....	3
3.1	Documents de référence.....	3
3.2	Définitions.....	3
3.3	Terminologie.....	3
4	Bonnes pratiques pour l'industrialisation de la production logicielle .....	4
4.1	Généricité du code .....	4
4.2	Indépendance logique / physique .....	4
4.3	Lisibilité du code et documentation.....	4
4.4	Prise en compte du contexte « sécuritaire ».....	4
4.5	Gestion des sources et des versions.....	4
4.5.1	Identification des applications .....	5
4.5.2	Identification des versions.....	5
4.5.3	Dépôt du code source.....	6
4.5.4	Livraison des codes sources.....	6
5	Référentiel général d'interopérabilité .....	6
6	Accessibilité (RGAA) .....	7

# 1 Objet

Ce document décline les bonnes pratiques pour l'industrialisation de la production logicielle au sein du ministère de la justice.

# 2 Domaine d'application

Ce document a vocation à être annexé aux CCTP publiés par le ministère.

# 3 Références, définitions et terminologie

## 3.1 Documents de référence

Référence	Version	Titre
<a href="https://references.modernisation.gouv.fr/rgaa-accessibilite/">https://references.modernisation.gouv.fr/rgaa-accessibilite/</a>	version 3.0	Référentiel Général d'Accessibilité pour les Administrations (RGAA)
<a href="http://references.modernisation.gouv.fr/interopabilite">http://references.modernisation.gouv.fr/interopabilite</a>	version 2.0	Référentiel Général d'Interopérabilité

## 3.2 Définitions

Terme	Définition

## 3.3 Terminologie

Acronyme	Définition
RAL	Résumé d'Architecture logicielle
RGAA	Référentiel Général d'Accessibilité pour les Administrations
RGI	Référentiel Général d'interopérabilité
RPVJ	Réseau Privé Virtuel Justice

## 4 Bonnes pratiques pour l'industrialisation de la production logicielle

### 4.1 Généricité du code

Favoriser autant que possible le recours à des éléments standards et généralisés au sein des équipes de développement, en utilisant autant que possible les principes d'héritage de composants standards, la création de procédures-types paramétrables, l'utilisation de librairies du marché, etc.

Placer les standards de développement et éléments de référence normalisés dans un environnement partagé par l'équipe de développement avec information régulière de son contenu et rappel de l'obligation de s'y référer en priorité, par le responsable du développement.

### 4.2 Indépendance logique / physique

Il est primordial de garantir une indépendance entre la vision logique de l'utilisateur sur les objets métier qu'il manipule et les choix physiques d'implémentation : l'évolution des choix techniques (noms d'objets physiques, persistance, localisation, structure) doit pouvoir se faire sans modifier la perception du métier à travers l'application.

Le développement doit donc intégrer une couche logicielle garantissant l'indépendance entre les objets logiques manipulés par les utilisateurs et les éléments physiques gérant les accès aux données (avec notamment prise en compte de la problématique des accès concurrents).

En particuliers :

- La lecture des données dans l'interface homme-machine est faite au moyen de vues sur la base de données et les actions sur les données (création, modification, suppression) sont faites au moyen d'appel de procédures, et non par accès direct aux tables.
- Regrouper dans un seul référentiel les composants de sources « bibliothèques » ayant le même nom complet ; le fait de déployer des micro-services n'autorise pas à dupliquer des bibliothèques pour chaque projet git créé.

### 4.3 Lisibilité du code et documentation

Tout élément applicatif est écrit en favorisant la lisibilité et la clarté du code.

Il est documenté individuellement dans un en-tête avec a minima : rôle de l'élément, contraintes particulières et limites d'utilisation, paramètres attendus en entrée et restitués en sortie, journal des évolutions, etc.

### 4.4 Prise en compte du contexte « sécuritaire »

Veiller à ce que :

- L'application n'implémente pas des fonctionnalités disponibles sur le serveur d'applications ;
- Le code livré soit exempt de tout code de débogage et de code mort ; Aucune vulnérabilité telle que le débordement de tampon, l'injection SQL, le « cross-site scripting » ou l'existence des portes dérobées doivent également être évitées ;
- Les erreurs systèmes et celles dues à un dysfonctionnement de l'application doivent être signalées de manière compréhensible à l'utilisateur avec un niveau de détails adapté

### 4.5 Gestion des sources et des versions

Le dépôt des codes sources (« repository ») et la gestion des versions permet : l'historisation de toutes les modifications, le maintien du code source tout au long du processus de développement, de travailler sur plusieurs versions en même temps, le travail collaboratif et la mutualisation des développements.

Tout élément applicatif est conservé dans un environnement sécurisé (sauvegardé, à accès contrôlé) et est associé à une ou plusieurs versions ou patch de l'application.

Il permet la reconstitution d'une version complète à tout moment sur n'importe quel environnement du Ministère de la Justice.

Le système de gestion des versions et de « repository » utilisé par le Ministère de la Justice est au format GIT.

L'outil de « gestion des dépendances et de build » utilisé par le Ministère de la Justice est MAVEN pour les développements JAVA.

Les solutions favorisant l'indépendance vis-à-vis du Système d'Exploitation (Windows et Linux) de déploiement sont retenues.

### 4.5.1 Identification des applications

Le système de gestion des versions permet de :

- Identifier pour un « projet » (ou Git) une seule application, au sein d'un « groupe » qui matérialise le système d'information ;
- Les « branches » au sein du projet ne sont pas là pour identifier des applications métier différentes mais bien des états d'avancements de développement d'une seule application.

### 4.5.2 Identification des versions

Toute livraison est faite via un bordereau de livraison qui fait apparaître la version, de binaires et de sources, à laquelle correspond cette livraison (quel que soit le nom donné au répertoire ou package de livraison).

L'identification des versions est effectuée :

- Selon les bonnes pratiques de nommage, de type VM.m.c ou VM.m.c-RCn (Majeur, mineur, complément, et en option ReleaseCandidate numéro). Les options de 4<sup>e</sup> digit sont à réserver pour les branches « features » hors branches de DEV ou MASTER). Pour être au plus proche de la Chaîne industrielle CODEO du Ministère de la Justice, les Tags identifiants des versions de recette doivent se limiter au 3 premiers digits ;
- Le changement de numéro de majeur correspond à un changement de l'API du logiciel. Le changement mineur correspond à un ajout de fonctionnalité sans changement d'API. Le complément correspond à de la correction de bug ;
- Les contextes de modifications techniques ou fonctionnels seront présents dans une Note de Contenu (Release Note) ;
- En les notant dans le RAL (résumé d'architecture logicielle) qui donne la correspondance, entre les versions d'application ou les versions des modules de cette application, et les positions dans le code source ; Une option consiste à fournir un fichier « sources.list » utilisé au ministère et dont le format peut être proposé aux équipes de développement, pour automatiser les matrices de compatibilités d'une version d'application avec tous les tags des projets git ou svn qui composent la version (y compris les batchs par exemple).
- Toute modification même mineure du code source livré entraîne un nouveau tag (i.e. le code source est modifié en permanence dans les branches de développement, seul celui livré doit obligatoirement être tagé) ;
- Tout tag Git existant ne doit être en aucune façon supprimé, déplacé, renommé, ou modifié dans son contenu, il doit rester immuable ;
- Tout projet java utilise maven comme outil de build ;
- Tout projet maven contient un super pom (ou pom.xml racine) dans lequel on déclarera une hiérarchie avec l'intégralité de ses modules fils ;
- Tout projet maven est identifiable et pour cela doit déclarer un groupId, un artifactId et un numéro de version unique dans le pom.xml racine ;
- Toute livraison de projet maven contient un numéro de version unique dans les pom.xml ;
- Le super-pom (ou pom.xml racine) du projet déclare obligatoirement un groupId, artifactId et un name qui reste constant au fil des livraisons de versions maven, afin que l'on puisse suivre un historique des versions ;
- Par extension, pour avoir une cohérence d'ensemble au sein du projet et dans le portefeuille du ministère de la justice, la version identifiée dans le nom du tag git est identique au numéro de version qui est déclaré dans le pom.xml (pour un projet maven). Ceci est un pré-requis pour des outils d'intégration continue (Jenkins) et des analyseurs de code (Cast, SonarQube, CheckMarx) ;
- Une version identifiée (tag) ne peut être livrée qu'une seule fois.
- Les références externes, notamment avec du code Open Source, sont autorisées. Mais si un projet intègre des composantes déjà développées avec d'autres systèmes de dépendance que Maven (comme Graddle), les architectes du ministère devront en valider la pertinence.
- Quelle que soit la norme utilisée, un ensemble de sources cohérents doit pouvoir être buildé ou analysé chez le partenaire ou au ministère sans action corrective du code source. Des profils doivent donc être intégrés aux fichiers d'environnement liés au dépendances (prestataire/ministère, dev/re7/prod, ...).

De plus, les bonnes pratiques de versionning impliquent un suivi rigoureux et régulier des actions de mise à jour dans le référentiel de code source en respectant les principes du « GitFlow » :

- La multiplication des branches de développement (« features ») jamais fusionnées est à bannir ;

- Les développeurs doivent valider et pousser (« commit » et « push ») régulièrement, dans des branches spécialisées (la branche « master » doit être protégée, elle est donc interdite aux commits des développeurs) ;
- Même en phase de développement, des tags (versions de type v0.m.n) doivent être créés régulièrement pour repérer des lots de sources, destinées à la lisibilité du code et à l'audit si besoin pour les longues périodes d'écritures.
- Les commits et les tags doivent être commentés (« message ») de façon pertinente, le libellé avec la version ne suffit pas.

### 4.5.3 Dépôt du code source

Le Ministère de la Justice récupère l'ensemble des éléments lui permettant de rejouer la construction du livrable (TAGs à partir desquels le livrable a été construit) via un client web de l'outil de gestion de versions :

- L'espace de dépôt de l'ensemble du code source (repository) est dédié au projet et organisé de telle sorte qu'il soit accessible par le Ministère de la Justice depuis son réseau (RPVJ) de manière permanente via un protocole de transfert hypertexte sécurisé (https) et qu'il soit techniquement possible d'en effectuer un export complet et exhaustif (y compris l'ensemble des scripts et batchs) au format du logiciel de gestion de versions utilisé (git).
- Cette copie de l'ensemble du code source pourra alors être importée via les outils du logiciel de gestion de versions utilisé sur une plate-forme dédiée du Ministère de la Justice où seront conservés les codes source des différentes versions applicatives livrées et/ou mises en production.

Dans la mesure où une mise en ligne des sources n'est pas possible rapidement (accord entre le Ministère de la Justice et le titulaire) :

- Une extraction via un export (SVN export) ou un clone (GIT clone) est réalisée dans une livraison compressée (.ZIP ou TAR.GZ) identifiée par le numéro de version que l'on retrouve sur le TAG de la branche principale d'intégration ;
- Une extraction de toutes les bibliothèques utilisées, publiques ou privées, devra être regroupée en mode système de fichier (extrait complet de repository Maven pour java, NuGet pour .Net, Satis pour PHP, etc...).
- L'outil est accessible par le Ministère de la Justice.

Le dépôt des codes sources ne doit en aucun cas être publié sur une plateforme publique telle que « GitHub ». La propriété intellectuelle du ministère prévaut sur la présomption de de code « open source » qu'un partenaire pourrait décider de façon unilatérale.

De la même manière, le code source d'applications aux langages interprétés (Php par exemple) ne doit pas apparaître sur des plateformes de tests hébergées sur des « clouds » publiques qui fournissent des machines virtuelles clé-en-main (Amazon par exemple).

### 4.5.4 Livraison des codes sources

Les codes sources livrés via les outils de versionning dédiés ne contiennent que du code source (pas de pollution avec des binaires, des bibliothèques, des archives, des documentations, etc...) et respectent les règles de livraison de versions avec TAG sur les branches d'intégration destinées à la recette, de type master (pas de trunk, pas de branche de dev).

Les sources livrées correspondent aux binaires présents dans le package identifié par cette version, et doivent donc pouvoir être recompilés pour aboutir à un binaire identique (ou liste de scripts identiques pour les langages scriptés).

Les scripts de recompilation de la version (ant, maven...) sont fournis.

La documentation détaillant la procédure à suivre pour recompiler est fournie.

## 5 Référentiel général d'interopérabilité

Le référentiel général d'interopérabilité fixe les obligations, recommandations et interdictions à respecter dans le cadre des échanges avec les autres administrations, collectivités territoriales et établissements publics à caractère administratif.

L'alignement au regard des contraintes du Référentiel Général d'interopérabilité (RGI) version V.2.0 présente un caractère obligatoire sur le plan réglementaire. Il s'applique aux administrations, collectivités territoriales et établissements publics à caractère administratif.

Le RGI est composé de trois volets : organisationnel, technique et sémantique.

## 6 Accessibilité (RGAA)

Le Référentiel Général d'Accessibilité pour les Administrations (RGAA) version V3.0 ou supérieure (cf. référence [R2]) a un impact structurant sur les applications et sur les choix technologiques de la partie IHM (Interface Homme-Machine).

Le niveau d'accessibilité attendu est le niveau RGAA double A (AA).

Le titulaire assure le contrôle de conformité du système d'information du Domaine X et accompagne le Ministère de la Justice dans la rédaction de la déclaration de conformité telle que décrite dans le Guide d'accompagnement du RGAA. Le titulaire détecte et justifie les contraintes techniques insurmontables pour l'accessibilité ou nécessitant des compromis dans la conception ou les choix opérés.