



AM 29/04/2024

SYSTÈME DE CAISSE ENREGISTREUSE – RÈGLEMENTATION TECHNIQUE –

DESCRIPTION DÉTAILLÉE DU FONCTIONNEMENT DU FISCAL DATA MODULE ET DE LA COMMUNICATION AVEC LE SYSTÈME DE CAISSE ET LE SERVICE CLOUD DU SPF FINANCES

VERSION 1.4 – 17/04/2026

CHANGELOG

Version	Date	Résumé
1.0	05/08/2024	Texte original
1.1	06/09/2024	Clarification: <ul style="list-style-type: none">- Connexion physique POS - FDM
1.2	20/09/2024	Correction: <ul style="list-style-type: none">- Remplacement table in chapter 3, section 3.3
1.3	20/05/2025	Clarification: <ul style="list-style-type: none">- Chapitre 4: lecture et prise de copie du buffer via backoffice
1.4	17/04/2026	Ajouts : <ul style="list-style-type: none">- 2.4 Signature numérique: déplacée ici depuis les descriptions détaillées POS Correction <ul style="list-style-type: none">- 6.1 verificationUrl: à un niveau supérieur

Introduction	1
ABRÉVIATIONS COURANTES.....	1
Chapitre 1 – Exigences techniques pour le Fiscal Data Module	2
1.1. Réglementations générales	2
1.1.1. Compteurs	2
1.1.2. Calcul de la TVA et de la base imposable.....	2
1.1.3. Paramètres du buffering et de personnalisations	3
1.1.4. Marquage CE.....	3
1.2. Exigences matérielles.....	3
1.2.1 Stockage.....	3
1.3. Interface utilisateur	3
1.4. Micrologiciel.....	4
Chapitre 2 – Communication avec le système de caisse connecté	5
2.1. Connexion physique.....	5
2.2. Service Graph QL.....	5
2.2.1. Général.....	6
2.2.2. Les mutations obligatoires	6
2.2.3. Contenu de la communication POS → FDM.....	7
2.2.4. Contenu de la communication FDM → POS.....	7
2.2.5. FDM → POS error handling.....	8
2.3. GraphQL schema pour la communication POS vers FDM.....	8
2.4. Signature numérique	8
2.4.1. JSON canonique pour le hachage reproductible.....	8
2.4.2. Le 'JSON enrichi' dont le contenu est signe numériquement	9
2.4.3. Certificat à utiliser pour la signature.....	10
Chapitre 3 – URL de vérification – QR-Code	11
3.1. Longueur fixe.....	11
3.2. Base-62 character set à utiliser	12
3.3. Bit stream	12
3.4. XOR Mask.....	13
3.5. Signature digitale.....	13
3.6. Signature bit teller et signature bits.....	14
3.7. FdmlId.....	14
3.8. totalCounter.....	14
3.9. Padding (remplissage).....	14
3.10. Valeurs de référence	15

3.12. Decoder.....	17
Chapitre 4 – Stockage temporaire des données	22
Chapitre 5 – Communication FDM ↔ Service Cloud du SPFFIN	23
5.1. Connexion et sécurité.....	23
5.1.1. Connexion.....	23
5.1.2. Sécurité.....	23
5.2. First Time Ever	23
5.3. API et protocole de la communication avec le service cloud du SPF finances	24
Chapitre 6 – Données de transaction à transmettre en ligne.....	25
6.1. MESSAGES JSON VERS LE SPF FIN	25

INTRODUCTION

Les descriptions détaillées reflètent de manière plus technique et détaillée les exigences techniques de l'arrêté ministériel (AM) du 29/04/2024 relatif aux aspects techniques en ce qui concerne la certification du Fiscal Data Module (FDM).

Ce document est la description technique détaillée concernant le fonctionnement du FDM, la communication avec un système de caisse certifié et la communication avec le service cloud du SPF Finances.

Pour le fonctionnement d'un système POS certifié, veuillez-vous référer à la description détaillée correspondante.

Ce document peut faire l'objet de modifications mineures, en raison d'éventuelles décisions réglementaires, afin de corriger d'éventuelles erreurs.

Des informations complémentaires peuvent être obtenues auprès du service concerné, Centre National de Recherche (CNR), division SCE à l'adresse secr.gksce@minfin.fed.be.

ABRÉVIATIONS COURANTES

AES	Advanced Encryption Standard
CBC	Cipher Block Chain
ECDSA	Elliptic Curve Digital Signature Algorithm
FDM	Fiscal Data Module
HTTP	Hyper Text Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON web token
LAN	Local Area Network
(m)TLS	(mutual) Transport Layer Security
NOP	No Operation
NTP	Network Time Protocol
PEM	Privacy Enhanced Mail
PKCS	Public Key Cryptography Standards
POS	Point Of Sale
QR code	Quick Response code
RTC	Real Time Clock
SCE	Système de Caisse Enregistreuse
SECP256r1	Specific Elliptic Curve Based on Prime Fields
TCP/IP	Transmission Control Protocol/Internet Protocol
UTC	Coordinated Universal Time
UTF	Unicode Transformation Format

CHAPITRE 1 – EXIGENCES TECHNIQUES POUR LE FISCAL DATA

MODULE

Référence AM: Titre II, Chapitre 2

1.1. RÉGLEMENTATIONS GÉNÉRALES

En principe, un FDM ne doit pas contenir de fonctionnalités autres que celles décrites dans l'AM. Il en va de même pour la communication entre le POS et le FDM et entre le FDM et le POS.

Si un fabricant souhaite inclure des fonctionnalités et/ou des communications supplémentaires, cela doit :

- Contribuer au bon fonctionnement (et donc à l'amélioration) de l'appareil ;
- Être documenté explicitement dans la demande de certification.

Toute fonctionnalité et/ou communication supplémentaire doit toujours être subordonnée aux fonctionnalités et à la communication (tant avec le POS qu'avec le service cloud du SPF Finances).

La réception et l'envoi de messages ainsi que leur traitement ne doivent jamais être interrompus pour cette fonctionnalité supplémentaire.

1.1.1. COMPTEURS

Les compteurs mentionnés à l'article 64 de l'AM commencent de nouveau à 1 s'ils ont atteint leur valeur maximale de 999999999.

1.1.2. CALCUL DE LA TVA ET DE LA BASE IMPOSABLE.

Exemple de calcul de la base imposable et du montant de la TVA (comme décrit à l'article 66) :

quantité	description	montant	code tva	code TVA	total	montant TVA non arrondi	montant TVA arrondi	base imposable
2	Food	41,50	B	A	10,00	1,73553719	1,74	8,26
2	Drinks	10,00	A	B	54,25	5,8125	5,81	48,44
1	Food	12,75	B	C	13,00	0,735849057	0,74	12,26
1	Take-away	13,00	C	D	15,00	0	0,00	15,00
5	Caution	2,00	X	X	2,00	0	0,00	2,00
1	Tabac	15,00	D					
		94,25			94,25		8,28	

Remarque importante : les taux de TVA peuvent changer. Grâce au service cloud du SPF Finances, le FDM recevra toujours les nouveaux taux et leur période de validité bien à l'avance. Si une période de réservation couvre deux jours calendaires (par exemple, du 30 juin 2024 à 16h00 au 1er juillet 2024 à 12h00), le FDM appliquera toujours les taux de TVA en vigueur à la date de réservation (bookingDate) communiquée par le POS. Dans l'exemple ci-dessus, si le taux de TVA du code B devait passer à 9 % le 1er juillet 2024, le FDM doit tout de même effectuer le calcul avec le taux en vigueur le 30 juin 2024 (bookingDate).

1.1.3. PARAMÈTRES DU BUFFERING ET DE PERSONNALISATIONS

Se référer au protocole de communication au chapitre 2.

1.1.4. MARQUAGE CE

Le fabricant doit s'assurer que ses produits répondent aux exigences de toutes les législations applicables. Il procède à une évaluation de la conformité en faisant appel - si la législation européenne l'exige - à un organisme notifié.

En outre, le fabricant tient un dossier technique sur ses produits et le présente, sur demande, aux autorités du pays où les produits sont commercialisés. Dans une déclaration de conformité, le fabricant déclare que ses produits sont conformes à l'ensemble de la législation applicable. Enfin, le fabricant appose le marquage CE sur le produit.

Plus d'info sur : [Questions fréquemment posées sur le marquage CE | SPF Economie \(fgov.be\)](#)

1.2. EXIGENCES MATÉRIELLES

Référence AM: Titre II, Chapitre 2, sous-chapitre 2.

1.2.1 STOCKAGE

Les stockages visés à l'article 76 et à l'article 81 de l'AM ne sont pas identiques. Il doit y avoir au moins 2 Go de capacité pour stocker (mettre en mémoire tampon) les données de transaction (JSON enrichi).

Ce stockage est totalement distinct de la mémoire interne décrite à l'article 76.

1.3. INTERFACE UTILISATEUR

Référence AM: Titre II, Chapitre 2, sous-chapitre 3.

Un nombre minimum d'indicateurs de santé doit être présent **sur l'appareil** lui-même. Ces indicateurs sont décrits à l'article 83 de l'AM. Le fabricant a le choix entre un display ou une LED.

Un manuel d'utilisation détaillé/troubleshooting sheet doit être mis à la disposition de l'utilisateur final, de l'installateur et du service compétent du SPF Finances.

L'accès au back-office du FDM doit être suffisamment protégé. Le fabricant le documente de manière détaillée dans sa demande de certification.

La configuration complète de la connexion réseau entre le POS et le service cloud du SPF Finances doit être possible via cet interface utilisateur.

L'article 85 de l'AM laisse le producteur libre de choisir entre :

- Une interface utilisateur sur l'appareil lui-même (écran) ;
- Un appareil connecté (ordinateur portable, tablette, etc.)

- Un POS lié (écran) ;
- Ou une combinaison de ce qui précède.

Le manuel d'utilisation et/ou d'installation décrit les procédures en détail. Ce manuel est également remis lors de la procédure de certification.

1.4. MICROLOGICIEL

Le micrologiciel porte toujours un numéro de version. Le certificat de conformité mentionne toujours ce numéro de version.

Le micrologiciel doit être capable de prendre en charge **toutes** les fonctionnalités demandées dans l'arrêté Ministériel et dans la présente description détaillée.

Par conséquent, le micrologiciel va :

- vérifier l'exactitude (conformément au schéma JSON et GraphQL) des messages reçus du POS ; les messages incorrects ne sont pas traités (signature numérique, calculs, incrémentation des compteurs, ...). Le POS est informé des erreurs via les codes d'erreur dans la réponse ;
- pour des messages valides, effectuer les calculs nécessaires, incrémenter les compteurs, réaliser les conversions et fournir la signature numérique ;
- stocker le contenu des messages signés et les préparer pour leur envoi au service cloud SPFFIN ;
- transmettre les données de transaction au service cloud du SPFFIN selon les modalités décrites plus loin ;
- être en mesure de recevoir les messages, comme le prévoit le protocole décrit plus loin, reçus du service cloud du SPFFIN, de les interpréter correctement et de prendre les mesures prévues ;

Les fabricants de FDM sont libres d'intégrer d'autres fonctionnalités. Toutefois, celles-ci ne peuvent pas affecter le bon fonctionnement fiscal de l'appareil et doivent contribuer au bon fonctionnement ou à la bonne utilisabilité du dispositif. Ces fonctionnalités sont décrites en détail dans la documentation fournie.

Le micrologiciel peut recevoir des mises à jour ou correctifs nécessaires ou imposés. Après approbation par le service compétent du SPF Finances, l'installation sur les appareils est effectuée par le fabricant ou son délégué, de manière sécurisée. Cette opération est décrite en détail dans la documentation fournie avec la demande de certification et fait partie du processus de certification.

CHAPITRE 2 – COMMUNICATION AVEC LE SYSTÈME DE CAISSE CONNECTÉ

2.1. CONNEXION PHYSIQUE

Le système de caisse et le FDM sont connectés par câble (LAN) ou sans fil (WiFi). La configuration se fait dans le back-office de la caisse et du FDM.

IMPORTANT : En théorie, la connexion sans fil peut également se faire via Internet. En cas de perte de connexion Internet, la connexion caisse - FDM sera automatiquement perdue et la caisse cessera d'enregistrer.

Le fabricant du FDM doit fournir suffisamment d'options pour permettre une sécurité adéquate de la connexion entre la caisse et le FDM. Des exemples typiques sont : HTTP avec token, HTTPS avec TLS avec certificat auto-signé, HTTPS avec mTLS utilisant un certificat auto-signé basé sur le certificat intermédiaire du fabricant de FDM.

La responsabilité ultime de l'installation et de la configuration correctes et sécurisées incombe au distributeur/installateur.

Le système de caisse et le FDM utilisent le protocole TCP/IP pour échanger leurs messages. Ces messages sont décrits plus en détail ci-dessous. D'autres messages ne sont autorisés que dans la mesure où ils n'affectent pas les fonctionnalités obligatoires et contribuent au bon fonctionnement de l'ensemble. Le FDM met son service GraphQL à disposition du POS via HTTP(S).

Le FDM est connecté à l'API cloud SPFFIN via Internet et reçoit également régulièrement des instructions via cette voie. Dans certains cas, voir la description détaillée correspondante, ces instructions peuvent entraîner l'affichage d'un message sur le système de caisse et/ou une intervention manuelle via le système de caisse.

Le FDM ne peut pas être connecté à des applications tierces. Seules les applications POS, les applications FODFIN et les applications du fabricant du FDM peuvent être connectées au FDM.

2.2. SERVICE GRAPHQL

L'échange de messages entre les POS et le FDM se fait par l'intermédiaire du service GraphQL du FDM.

La norme GraphQL du 27 octobre 2021 est entièrement respectée ici.

De plus amples informations sont disponibles à l'adresse suivante : [Release October 2021 - graphql/graphql-spec - GitHub](#)

Les requêtes sont envoyées par le POS au service GraphQL du FDM. Les données JSON sont utilisées comme payload pour l'envoi des données d'événement. Ce payload est envoyé au chemin **"/graphql"** sur le FDM avec le verbe HTTP "POST" et Content-Type : application/json.

Tous les objets JSON sont conformes à la norme RFC 8259.

Les champs de type chaîne (string) ne contiennent **jamais** d'espaces blancs en début ou en fin de chaîne.

Les données envoyées comprennent

- des données de transaction (events P et N) ;
- des données financières (event F) ;
- des données sociales (event S) ;

- les copies (event C)
- les events de formation (event T)
- les factures (event I)
- les rapports (event R).

Ces objets et leurs conditions sont décrits plus en détail dans le schéma GraphQL complet.

2.2.1. GÉNÉRAL

Les events suivants ont été déterminés dans l'Arrêté ministériel :

- NORMAL (label N);
- PRO FORMA (label P);
- TRAINING (label T);
- COPY (label C);
- FINANCIAL (label F);
- SOCIAL (label S);
- INVOICE (label I);
- REPORT (label R).

Ces events sont tous envoyés au FDM comme protection contre la manipulation des données afin d'être signés numériquement. À cette fin, diverses mutations sont mises à disposition par le service GraphQL du FDM. Ces mutations peuvent être utilisées pour remplir correctement la structure JSON des enregistrements sur le FDM comme décrit ci-dessous. La structure JSON du FDM peut contenir tout type d'événement ; via les différentes mutations, le FDM n'accepte pour chaque type d'événement que les champs et les objets autorisés.

2.2.2. LES MUTATIONS OBLIGATOIRES

Les mutations suivantes sont définies.

Pour l'événement S:

- signWorkIn
- signWorkOut

Pour l'événement I:

- signInvoice

Pour l'événement N:

- signSale

Pour l'événement P:

- signCostCenterChange
- signOrder
- signPreBill

Pour l'évent F:

- signMoneyInOut
- signDrawerOpen
- signPaymentCorrection

Pour l'évent R:

- signReportTurnoverX
- signReportTurnoverZ
- signReportUserX
- signReportUserZ

Pour l'évent C:

- signCopy

2.2.3. CONTENU DE LA COMMUNICATION POS → FDM

Pour cette section, veuillez consulter la **DESCRIPTION DÉTAILLÉE POS**, également mise à disposition via : www.systemedecaisseenregistreuse.be/fr/sce-2-0.

Règles de communication

Afin de garantir un traitement correct des mutations et de les distinguer de manière unique, les champs clés suivants sont déterminés :

- **posId**
- **posDateTime**
- **terminalId**
- **eventLabel**
- **posFiscalTicketNo**

Lorsqu'un POS transmet une mutation au FDM où ces 5 champs clés sont identiques à ceux envoyés dans un message précédent, le FDM doit vérifier s'il s'agit ou non d'une mutation à laquelle il a déjà été répondu (y compris en augmentant les compteurs concernés dans le FDM).

Les situations suivantes peuvent se présenter :

1. La combinaison des champs clés est différente : le FDM traite cette mutation de manière normale.
2. Le contenu de la mutation est totalement identique : le FDM traite cette mutation comme un 'double' envoi et renvoie dans la réponse les mêmes données que dans la mutation originale; le FDM n'augmente pas non plus ses compteurs ; cette transaction en double n'est pas suivie dans le FDM.
3. La combinaison des champs clés est identique, mais les autres valeurs de la mutation sont différentes de celles de la mutation précédemment traitée : le FDM refuse de traiter cette mutation et envoie un code d'erreur dans sa réponse.

2.2.4. CONTENU DE LA COMMUNICATION FDM → POS

Pour cette section, veuillez consulter la **DESCRIPTION DÉTAILLÉE POS**, également mise à disposition via : www.systemedecaisseenregistreuse.be/fr/sce-2-0.

2.2.5. FDM → POS ERROR HANDLING

Pour cette section, veuillez consulter la **DESCRIPTION DÉTAILLÉE POS**, également mise à disposition via : www.systemedecaisseenregistreuse.be/fr/sce-2-0.

Note générale concernant le traitement des erreurs.

Certaines erreurs et avertissements sont affichés à l'utilisateur via l'interface utilisateur du POS. En plus des dispositions ci-dessus, le fabricant du FDM fournit à l'utilisateur les informations supplémentaires nécessaires afin qu'il puisse prendre les mesures appropriées pour résoudre l'erreur.

Ceci est décrit en détail dans la documentation fournie avec la demande de certification.

2.3. GRAPHQL SCHEMA POUR LA COMMUNICATION POS VERS FDM

Pour cette section, veuillez consulter la **DESCRIPTION DÉTAILLÉE POS**, également mise à disposition via : www.systemedecaisseenregistreuse.be/fr/sce-2-0.

2.4. SIGNATURE NUMÉRIQUE

La signature est apposée sur le « JSON enrichi » après application du réencodage JSON. Cela permet de garantir la reproductibilité de la signature.

2.4.1. JSON canonique pour le hachage reproductible

La flexibilité offerte par JSON permet d'encoder les mêmes informations de différentes manières. Pour obtenir toujours la même signature pour les mêmes données sources, un certain nombre de règles sont appliquées. Ces règles conduisent à un JSON canonique.

Les règles suivantes doivent être strictement appliquées par le FDM aux champs JSON mentionnés plus loin au point 2.4.2.

- Les **nombres** ne peuvent contenir que les caractères 0 à 9, le signe moins (-) et un point décimal. Cela signifie, entre autres, que les valeurs contenues dans les notes scientifiques doivent être recodées sous leur forme la plus élémentaire. Les zéros après une virgule ne sont pas autorisés.
- Tous les **espaces**, comme décrit dans les spécifications JSON, sont supprimés, aucun espace n'est ajouté.
- Les caractères avec un **point de code > U+001F et < U+007F** ne sont **jamais** échappés, à l'exception des points de code **U+0022** (guillemets doubles) et **U+005C** (barre oblique inverse) qui sont toujours échappés conformément aux spécifications JSON. Ces deux-là doivent être échappés avec la note la plus courte possible (\' et \\ respectivement).
- Les caractères avec un **point de code <U+0020 ou > U+007E** sont **toujours** échappés et toujours dans la notation la plus courte possible. Backspace, formfeed, linefeed, carriage return et horizontale tab ont de telles notations courtes (\b, \f, \n, \r et \t respectivement).
- Lors de l'échappement d'un point de code sous la forme d'un ou plusieurs ensembles de 4 caractères hexadécimaux, les caractères hexadécimaux sont toujours en **majuscule** (par exemple : \uA AFF).

- Le décodeur JSON doit respecter l'interprétation **stricte** des spécifications JSON. Par exemple : les littéraux vrai, faux, nul sont toujours en minuscules et sans guillemets, les noms des couples nom/valeur dans les objets doivent toujours être entre guillemets, ...
- Les **paires nom/valeur** de chaque objet en JSON sont **triées**, par ordre croissant, par la valeur numérique des points de code dans le nom. Il est supposé, à titre de bonne pratique, qu'aucun nom double n'est utilisé, car il n'existe pas de norme JSON à cet égard.

2.4.2. Le 'JSON enrichi' dont le contenu est signé numériquement

L'objet JSON 'enrichi' (**enrichedEventData**) est constitué des valeurs et objets ci-dessous, selon qu'ils ont été utilisés ou non dans le message.

Les champs optionnels contenant une valeur null ou les array's optionnels vides ne sont pas reprises dans l'objet et ne sont donc pas reprises dans la signature non plus.

```
{
  language
  posId
  vatNo
  estNo
  terminalId
  deviceId
  posDateTime
  posFiscalTicketNo
  ticketMedium
  eventOperation
  copyOfEvent
  employeeId
  customerVatNo
  invoiceNo
  fdmRefs
  bookingPeriodId
  bookingDate
  costCenter
  transfer
  transaction
  vatCalc
  financials
  drawer
  reportNo
  reportBookingDate
  posDevices
  fdmDevices
  turnover
  users
  fdmSwVersion
  posSwVersion
  bufferCapacityUsed
  fdmId
  fdmDateTime
  eventLabel
  eventCounter
  totalCounter
}
```

La valeur copyOfEvent est déterminée et ajoutée par le FDM.

copyOfEvent (enum, obligatoire).

Le eventLabel détecté par le FDM lors de la réception des messages du POS. Valeurs à utiliser à partir de l'enum EventLabel (voir description précédente).

2.4.3. Certificat à utiliser pour la signature

La signature doit être apposée à l'aide du certificat matériel dont les détails sont décrits dans la description détaillée du fonctionnement et de la communication entre le module de données fiscales et le service cloud du SPF.

CHAPITRE 3 – URL DE VÉRIFICATION – QR-CODE

Le POS doit imprimer un code QR sur son ticket de caisse TVA. À cette fin, le FDM crée l'URL. Cet URL se compose de deux parties :

- le préfixe fixe : <https://www.gs2.be>
- la valeur codée

Le préfixe est fixe et est déterminé par le SPF Finances.

La valeur codée est entièrement calculée par le FDM et est unique pour chaque event N. Cette valeur contient :

- Une indication du nombre de bits de la digitalSignature qui ont été inclus dans le calcul ;
- Les bits de la digitalSignature (avec un maximum de 5 octets) ;
- Le fdmId ;
- Le compteur total des events ;
- Pas ou plus de bits de remplissage

Exemple:



Préfixe:
<https://www.gs2.be/>

Valeur codée:
viXnmc1vvqGaMAS8MJV

3.1. LONGUEUR FIXE

La valeur codée est **toujours** d'une longueur de 19base-62 caractères.

Étant donné que les caractères en base-62 peuvent être de cinq ou six bits, il est possible qu'un remplissage soit nécessaire. D'autre part, il est également possible que la valeur codée dépasse 19 caractères. Dans ce cas, la quantité de données dans la valeur codée doit être réduite.

Par défaut, l'encodeur inclura 40 bits de la digitalSignature ; si nécessaire, cela peut être réduit à 36, 32 ou 28 bits. Bien entendu, l'encodeur doit toujours inclure le nombre maximal de bits possible de la digitalSignature.

3.2. BASE-62 CHARACTER SET À UTILISER

Decimal	Binary	Base62
0	00000	0
1	11111	1
2	000010	2
3	000011	3
4	000100	4
5	000101	5
6	000110	6
7	000111	7
8	001000	8
9	001001	9
10	001010	A
11	001011	B
12	001100	C
13	001101	D
14	001110	E
15	001111	F

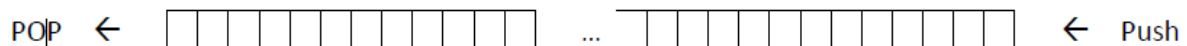
Decimal	Binary	Base62
16	010000	G
17	010001	H
18	010010	I
19	010011	J
20	010100	K
21	010101	L
22	010110	M
23	010111	N
24	011000	O
25	011001	P
26	011010	Q
27	011011	R
28	011100	S
29	011101	T
30	011110	U
31	011111	V

Decimal	Binary	Base62
32	100000	W
33	100001	X
34	100010	Y
35	100011	Z
36	100100	a
37	100101	b
38	100110	c
39	100111	d
40	101000	e
41	101001	f
42	101010	g
43	101011	h
44	101100	i
45	101101	j
46	101110	k
47	101111	l

Decimal	Binary	Base62
48	110000	m
49	110001	n
50	110010	o
51	110011	p
52	110100	q
53	110101	r
54	110110	s
55	110111	t
56	111000	u
57	111001	v
58	111010	w
59	111011	x
60	111100	y
61	111101	z

1.3. BIT STREAM

Aussi bien les routines d'encodage que de décodage utilisent un tampon de 64 bits fonctionnant comme un flux de bits. Les bits sont insérés dans le flux du moins significatif au plus significatif et sont traités dans le flux du plus significatif au moins significatif.



La valeur la plus longue à insérer dans le flux est de 41 bits. Le traitement du flux de bits ne commence qu'après l'insertion complète de chaque champ, afin d'éviter tout dépassement. Les bits traités sont immédiatement retirés du flux.

Un compteur suit en continu le nombre de bits insérés dans le flux. Le nombre de bits traités diminue le compteur à chaque fois.

Dès que les cinq bits les plus anciens représentent 00000 ou 11111, ils sont traités comme une unité de cinq bits ; toutes les autres valeurs sont traitées comme une unité de six bits.

L'encodeur insère la représentation binaire des données originales dans le flux de bits et utilise le tableau de la section 3.2 pour déterminer le caractère base-62 correct correspondant aux bits qu'il traite.

Le décodeur utilise ensuite le caractère base-62 pour déterminer combien de bits et selon quel motif les insérer dans le flux, et il reconstruit ainsi les données codées originales à partir des bits qu'il traite.

3.4. XOR MASK

Pour obtenir une bonne répartition des valeurs codées, un masque XOR est appliqué aux valeurs de `fdmId` et de `totalCounter`.

Sans ce masque, les valeurs récurrentes de `fdmId` et l'incrémentation de `totalCounter` pourraient former un motif reconnaissable dans les valeurs codées.

Le masque est généré à partir des deux premiers octets de la `digitalSignature` (appelés ci-après B1 et B2).

```
set mask equal to B2
shift the bits in the mask to the left by 8 positions
add B1 to mask
shift the bits in the mask to the left by 8 positions
add the inverse of B1 to mask
shift the bits in the mask to the left by 8 positions
add the inverse of B2 to mask
shift the bits in the mask to the left by 8 positions
add B1 to mask
shift the bits in the mask to the left by 8 positions
add B2 to mask
```

Parmi les 48 bits du masque obtenu, les 41 bits les moins significatifs sont utilisés comme masque pour le `fdmId`. Pour le `totalCounter`, les 30 bits les moins significatifs de l'inverse de ce masque de 48 bits sont utilisés comme masque.

3.5. SIGNATURE DIGITALE

Jusqu'à cinq octets de la `digitalSignature` sont inclus dans la valeur codée. Les premier, deuxième et troisième octets de la `digitalSignature` sont **toujours** inclus. Les quatrième et cinquième octets peuvent être entièrement ou partiellement omis si la valeur codée dépasse 19 caractères en base 62.

Ils sont inclus ou omis par **demi-octets** selon la séquence 40, 36, 32, 28, comme décrit ci-dessous.

B1	B2	B3	B4		B5	
Always	Always	Always	Always (28)	>= 32	>= 36	= 40
0x00 – 0xFF	0x00 – 0xFF	0x00 – 0xFF	0x0 – 0xF	0x0 – 0xF	0x0 – 0xF	0x0 – 0xF

3.6. SIGNATURE BIT TELLER ET SIGNATURE BITS

Le compteur de bits de la signature est le **premier champ** à être inséré dans le flux de bits.

Ce compteur indique le nombre de bits composant la signature numérique.

Ce champ contient toujours **deux** bits.

1	1	40 bits
1	0	36 bits
0	1	32 bits
0	0	28 bits

Le champ du compteur de bits de la signature est immédiatement suivi par le nombre correspondant de bits de la signature (**deuxième champ**).

3.7. FDMID

Le **troisième champ** à insérer dans le flux de bits est le fdmId. Il s'agit d'une chaîne composée de trois caractères allant de A à Z, suivis de huit chiffres allant de 0 à 9.

Les trois caractères sont convertis en valeurs intégrales (c1, c2, c3) comprises entre 0 et 25, où A = 0, B = 1, C = 2, et ainsi de suite.

Les huit chiffres de la chaîne sont transformés en une valeur intégrale (ms). Enfin, ces quatre valeurs sont combinées en une seule valeur intégrale de 41 bits en utilisant la formule suivante $v = ((((((ms * 26) + c3) * 26) + c2) * 26) + c1)$

Avant que cette valeur ne soit insérée dans le flux de bits, le masque XOR de 41 bits est d'abord appliqué.

3.8. TOTALCOUNTER

Le **quatrième champ** à insérer dans le flux de bits est le totalCounter. Il s'agit d'une valeur intégrale de 30 bits.

Avant que cette valeur ne soit insérée dans le flux de bits, le masque XOR de 41 bits lui est d'abord appliqué.

3.9. PADDING (REPLISSAGE)

L'éventuel padding appliqué a deux objectifs principaux :

- Il garantit le bon fonctionnement du flux de bits et que tous les bits sont codés en valeurs base 62 ;
- Il assure que le résultat final a une longueur de 19 caractères en base 62.

Les bits de remplissage ont toujours la valeur 0. Un remplissage non nul n'est pas autorisé, et les valeurs codées contenant de tels remplissages doivent être considérées comme invalides.

3.10. VALEURS DE RÉFÉRENCE

Les valeurs ci-dessous peuvent être utilisées comme référence pour vérifier les implémentations d'encodage et de décodage.

FDM serial	TotalCounter	Original signature bytes (hex)					Encoded	Signature bits
FDM01000001	1	9B	21	C7	09	BF	viXnmc1vvqGaMAS8MJV	40
FDM01000001	2	26	8D	D5	E5	F4	oQDrUNqaEVtV0cdZTbm	40
FDM01000001	3	B4	8B	E6	23	67	xIBvYDdswxqDH0rYqjt	40
FDM01000001	999999997	F8	4B	C9	FF	27	lXBoVylc8b0cu0MZd98	36
FDM01000001	999999998	20	26	5B	00	3C	Y0JB00CDvAB1ergy5YS	36
FDM01000001	999999999	75	E1	F9	FD	97	tNX1FxBt2UwgjADUqFX	40
AAA01000001	1	42	A6	98	AD	4A	qAccArA8jNY1mE3fhr0	40
AAA01000001	999999999	B1	5E	58	9A	4C	x5UM9fCMYqUX1s9n8TU	40
AAA99999999	1	5B	4C	86	12	4C	rjCXX9CuSrM70KSJAIo	40
AAA99999999	999999999	F7	40	CD	C4	62	lT0Pk8r1Fg2K60UrWe0	36
ZZZ01000001	1	96	E1	D0	9B	C5	vRXq915oB1WvMkNuMaV	40
ZZZ01000001	999999999	74	C6	C6	CC	49	tJ6nin9EyxIEyISk8MC	40
ZZZ99999999	1	45	94	D8	4D	B1	qMKs4snxi0TE9h6bBfg	40
ZZZ99999999	999999999	0D	F3	38	D8	B5	mtpEDYrIkG9rGPgqdVc	40
XYZ98123456	33751	99	F2	2F	B2	CB	vdoBxBBXF50zRorvSkq	40
XYZ98123456	33766	D4	A0	48	B4	45	zIWIBH5eL2nqnMhGL5o	40
XYZ98123456	33788	35	13	34	9A	01	pKJD9e0gL2q4U0s4qaG	40

3.11. Encoder

Cette implémentation part du principe que toutes les entrées sont des valeurs valides.

```
public static byte[] Encode(string fdmSerial, long totalCounter, byte[] digitalSignature)
{
    // State machine states:
    // 0. initialize
    // 1. push signature bitcount modifier
    // 2. push signature bits, apply bitcount modifier
    // 3. push FDM manufacturer, model, and serial number
    // 4. push TotalCounter
    // 5. flush the stream by adding padding if need be
    // 6. verify length of the output (return the result)

    byte[] bytes = new byte[19];
    long bitStream = 0;
    long mask = digitalSignature[1];
    mask = (mask << 8) + digitalSignature[0];
    mask = (mask << 8) + ~digitalSignature[0];
    mask = (mask << 8) + ~digitalSignature[1];
    mask = (mask << 8) + digitalSignature[0];
    mask = (mask << 8) + digitalSignature[1];
    for (int state = 0, signatureBitCountModifier = 4, byteCount = 0, streamLen = 0; state < 7 &&
signatureBitCountModifier > 0; state++)
    {
        switch (state)
        {
            case 0:
```

```
bitStream = 0;
streamLen = 0;
byteCount = 0;
break;
```

case 1:

```
bitStream = (bitStream << 2) | (byte)(signatureBitCountModifier - 1);
streamLen += 2;
break;
```

case 2:

```
bitStream = (bitStream << 8) | digitalSignature[0];
bitStream = (bitStream << 8) | digitalSignature[1];
bitStream = (bitStream << 8) | digitalSignature[2];
bitStream = (bitStream << 8) | digitalSignature[3];
bitStream = (bitStream << 8) | digitalSignature[4];
streamLen += 40;
```

```
int undo = (4 * (4 - signatureBitCountModifier));
bitStream = (bitStream >> undo);
streamLen -= undo;
break;
```

case 3:

```
int c1 = fdmSerial[0] - 65;
int c2 = fdmSerial[1] - 65;
int c3 = fdmSerial[2] - 65;
long ms = int.Parse(fdmSerial.Substring(3, 8));
long mask41 = mask & (((long)1 << 41) - 1);
long v = (((((ms * 26) + c3) * 26) + c2) * 26) + c1) ^ mask41;
bitStream = (bitStream << 41) | v;
streamLen += 41;
break;
```

case 4:

```
long mask30 = (~mask) & (((long)1 << 30) - 1);
bitStream = (bitStream << 30) | (totalCounter ^ mask30);
streamLen += 30;
break;
```

case 5:

```
if (0 != streamLen)
{
    int n = 6 - streamLen;
    bitStream = bitStream << n;
    streamLen += n;
}
break;
```

case 6:

```
while (bytes.Length > byteCount)
bytes[byteCount++] = 48; /* padding with "0" */
```

```

        return bytes;
    }
    /* try to read from the bit stream */while (6 <= streamLen)
    {
        if (bytes.Length <= byteCount)
        {
            /* we are ending up with too many characters, drop half a signature byte and start
            over */
            signatureBitCountModifier--;
            state = -1;
            break;
        }

        int read = 6;
        int bits = (int)((bitStream >> (streamLen - read)) & 63);
        switch (bits)
        {
            case 0:
                read = 5;
                break;

            case 1:
                bits = 0;
                read = 5;
                break;

            case 62:
            case 63:
                bits = 1;
                read = 5;
                break;
        }
        bytes[byteCount++] = (byte)(bits + ((10 > bits) ? 48 : (35 < bits) ?
            61 : 55));
        streamLen -= read;
        bitStream &= (((long)1) << streamLen) - 1;
    }
}
/* if we got here the encoding failed */return null;
}

```

3.12. DECODER

Cette implémentation suppose que le paramètre bytes est un array d'une longueur égale ou supérieure à 19.

```

public static bool Decode(byte[] bytes)
{
    // State machine states:
    // 0. get signatureBitCount
    // 1. get signature byte #1

```

```

// 2. get signature byte #2
// 3. get signature byte #3
// 4. get signature byte #4 MSB (according to the signatureBitCount)
// 5. get signature byte #4 LSB (according to the signatureBitCount)
// 6. get signature byte #5 MSB (according to the signatureBitCount)
// 7. get signature byte #5 LSB (according to the signatureBitCount)
// 8. get FDM manufacturer, model, and serial number
// 9. get TotalCounter
// 10. verify padding (must be zeroes)

int state = 0;
long bitStream = 0;
int streamLen = 0;
int nextByteIndex = 0;
long mask = 0;

/* output variables */
StringBuilder fdm = new StringBuilder();
long totalCounter = 0;
byte[] signatureBytes = new byte[5];
int signatureBitCount = 40;

while (state <= 10)
{
    if (nextByteIndex < bytes.Length)
    {
        int b = bytes[nextByteIndex];
        bitStream &= (((long)1) << streamLen) - 1;

        /* convert the byte to the range (-1, 63), where -1 indicates illegal character */
        b -= (48 <= b && 57 >= b) ? 48 : (65 <= b && 90 >= b) ?
            55 : (97 <= b && 122 >= b) ? 61 : b + 1;

        switch (b)
        {
            case -1:
                return false; /* invalid character in input */

            case 0:
                bitStream = (bitStream << 5);
                streamLen += 5;
                break;

            case 1:
                bitStream = (bitStream << 5) | 31;
                streamLen += 5;
                break;

            default:
                bitStream = (bitStream << 6) | (byte)b;
                streamLen += 6;
                break;
        }
    }
}

```

```

    }
    nextByteIndex++;
}
else if (nextByteIndex++ > bytes.Length)
    return false; /* we ran out of characters but the state machine is still waiting for more bits */

switch (state)
{
    case 0:
        if (2 <= streamLen)
        {
            int modifier = (int)((bitStream >> (streamLen - 2)) & 3) + 1;
            signatureBitCount -= (4 * (4 - modifier));
            streamLen -= 2;
            state++;
        }

        break;

    case 1:
    case 2:
    case 3:
        if (8 <= streamLen)
        {
            signatureBytes[state - 1] = (byte)(bitStream >> (streamLen - 8));
            streamLen -= 8;
            state++;
        }
        break;

    case 4:
        if (4 <= streamLen)
        {
            if (28 <= signatureBitCount)
            {
                signatureBytes[state - 1] |=
                    (byte)((bitStream >> (streamLen - 4)) & 15) << 4;
                streamLen -= 4;
            }
            state++;
        }
        break;

    case 5:
        if (4 <= streamLen)
        {
            if (32 <= signatureBitCount)
            {
                signatureBytes[state - 2] |=
                    (byte)((bitStream >> (streamLen - 4)) & 15);
                streamLen -= 4;
            }
            state++;
        }

```

```

    }
    break;

case 6:
    if (4 <= streamLen)
    {
        if (36 <= signatureBitCount)
        {
            signatureBytes[state - 2] |=
                (byte)((bitStream >> (streamLen - 4)) & 15) << 4);
            streamLen -= 4;
        }
        state++;
    }
    break;

case 7:
    if (4 <= streamLen)
    {
        if (40 <= signatureBitCount)
        {
            signatureBytes[state - 3] |=
                (byte)((bitStream >> (streamLen - 4)) & 15);
            streamLen -= 4;
        }
        state++;
    }
    break;

case 8:
    if (41 <= streamLen)
    {
        mask = signatureBytes[1];
        mask = (mask << 8) + signatureBytes[0];
        mask = (mask << 8) + ~signatureBytes[0];
        mask = (mask << 8) + ~signatureBytes[1];
        mask = (mask << 8) + signatureBytes[0];
        mask = (mask << 8) + signatureBytes[1];

        long mask41 = mask & ((long)1 << 41) - 1;
        long v = ((bitStream >> (streamLen - 41)) & 0xFFFFFFFF) ^ mask41;
        fdm.Append((char)((v % 26) + 65));
        fdm.Append((char)((v / 26 % 26) + 65));
        fdm.Append((char)((v / 676 % 26) + 65));
        fdm.Append((v / 17576).ToString("00000000"));
        streamLen -= 41;
        state++;
    }
    break;

case 9:
    if (30 <= streamLen)

```

```

    {
        long mask30 = (~mask) & ((long)1 << 30) - 1;
        totalCounter =
            ((long)((bitStream >> (streamLen - 30)) & 0xFFFFFFFF) ^
            mask30;
        streamLen -= 30;
        state++;
    }
    break;

case 10:
    if (0 != (bitStream & ((long)1 << streamLen) - 1))
        return false; /* non-zero padding encountered */
    return true; /* output variables are set */
}
return false;
}
}

```

CHAPITRE 4 – STOCKAGE TEMPORAIRE DES DONNÉES

Le FDM dispose d'une mémoire de stockage de données de minimum 2 Go. Cela permet au FDM de stocker temporairement les messages « enriched JSON » avant de les envoyer au service cloud du SPFFIN.

Ces JSON enrichis, complétés par quelques champs supplémentaires (voir plus loin), constituent la charge utile (payload) des messages qui seront envoyés vers ws_reg. Le FDM veille à ce que ces messages ne puissent être ni supprimés ni modifiés.

Les messages stockés ne peuvent être copiés ou consultés que via le back office.

Le cas échéant qu'il n'a pas moyen d'avoir un réseau internet disponible, cette consultation et prise de copie doit être disponible via le backoffice.

Les messages « enriched JSON » transmis ne peuvent être supprimés qu'après leur envoi, conformément aux dispositions du chapitre 5.

CHAPITRE 5 – COMMUNICATION FDM ↔ SERVICE CLOUD DU SPFFIN

5.1. CONNEXION ET SÉCURITÉ

5.1.1. CONNEXION

La connexion via l'internet est configurée via le back-office du FDM. Ce back-office est spécifique au producteur et l'interface utilisateur peut être différente. Au minimum, le back-office offre les options de configuration suivantes :

- l'url du service cloud du SPFFIN ;
- l'url du time server.

5.1.2. SÉCURITÉ

Chaque communication est sécurisée par l'utilisation du :

- certificat d'authentification
- protocole mTLS qui crypte les envois.

5.2. FIRST TIME EVER

Conformément aux dispositions de l'arrêté ministériel du 29/04/2024, tant le système de caisse que le Fiscal Data Module sont enregistrés par les acteurs impliqués dans les différentes phases (production, livraison) dans l'e-service en ligne SCE du SPF Finances.

Cela détermine, au niveau de l'entreprise et du secteur, quel FDM est lié à quel système de caisse.

Lors de la mise en service du SCE, la caisse et le FDM sont liés entre eux. Avec une mise en route correcte (responsabilité unique de l'entreprise de restauration et de son(ses) fournisseur(s)), le FDM prendra contact avec le service cloud du SPF Finances via une connexion internet sécurisée. Le service cloud SPFFIN :

1. vérifie le numéro de série du FDM, attribue le certificat d'authentification unique et personnalisé et l'envoie via la connexion sécurisée au FDM, qui le stocke dans l'environnement sécurisé (art. 77 de cette décision ministérielle) ;
2. envoie simultanément un paquet d'initialisation (aperçu des taux de TVA applicables, réglages de fréquence personnalisés , etc.);

La description détaillée complète du protocole incluse dans le document FPSFIN_API_PROTOCOL_FDM_FINCLOUD, sera mis à disposition via : www.systemedecaisseenregistreuse.be/fr/sce-2-0.

5.3. API ET PROTOCOLE DE LA COMMUNICATION AVEC LE SERVICE CLOUD DU SPF FINANCES

Référence AM: Titre II, Chapitre 2, sous-section 4.

Le FDM communique avec les serveurs (cloud) SPFFIN via le protocole inclus dans le document FPSFIN_API_PROTOCOL_FDM_FINCLOUD, qui sera mis à disposition via : www.systemedecaisseenregistreuse.be/fr/sce-2-0.

Aucune communication autre que celle-ci n'est autorisée.

CHAPITRE 6 – DONNÉES DE TRANSACTION À TRANSMETTRE EN LIGNE

En fonction de la fréquence de transmission configurée, le FDM transmet les paquets contenant les données de transaction du FDM au service cloud SPFFIN. De tels packages contiennent au moins le contenu d'un event complet ou d'un NOP.

Le service cloud SPFFIN confirme (ou infirme) la bonne réception des paquets. Le FDM conserve ces paquets pendant 10 jours supplémentaires, après quoi ils peuvent être supprimés du buffer.

Ces paquets sont conservés par le service cloud du SPFFIN pendant au moins 3 ans.

Ce stockage n'affecte pas les obligations légales fiscales de conservation et de présentation des données originales sur le système de caisse (articles 315bis et 315ter du CIR92 et articles 60 et 63 du CTVA).

Les données autres que celles incluses dans l'Arrêté ministériel ne peuvent et ne doivent pas être transmises via cette connexion.

6.1. MESSAGES JSON VERS LE SPF FIN

Les messages JSON qui sont transmis vers les serveurs SPFFIN via `ws_reg` contient les arrays, objets et valeurs tels que transmis au FDM par le système de caisse et tels que créés par le FDM lui-même. Nous appelons également cela le JSON enrichi.

```
{
```

unsentEventCount (numerique, obligatoire)

Nombre de messages non envoyés dans la mémoire tampon. Toujours 0 en cas de No Operation (NOP).

fdmId (string, obligatoire)

Le numéro de série du FDM.

events (array, obligatoire)

Array d'events enrichis. Obligatoire, dans le cas d'un NOP, l'array est vide.

```
[
```

```
{
```

enrichedEventData (object, obligatoire)

Objet contenant des données d'events provenant du POS, complétées par des calculs provenant du FDM. Les champs inclus dans cet objet sont décrits dans les Descriptions Détaillées POS.

digitalSignature (string, obligatoire)

Signature numérique en base64 basée sur le recodage JSON canonique de `enrichedEventData`.

shortSignature (string, conditionnel)

Le checksum SHA-1 en hexadécimales de la signature numérique en bytes. Calculé uniquement pour l'event N.

verificationUrl (string, conditionnel)

Cette URL est générée par le FDM sur la base d'un préfixe fixe et d'une valeur encodée, afin d'être imprimée sous forme de code QR sur le ticket de caisse TVA. Calculé uniquement pour l'événement N.

fdmLocalisation (objet, obligatoire)

}

Dans des cas exceptionnels, lorsque des erreurs se produisent lors de la lecture d'une transaction, cet objet peut être remplacé par :

{

rawCounter (numérique, obligatoire)

Ce compteur est augmenté chaque fois qu'une transaction corrompue est transmise.

rawBytes (string, obligatoire)

Les bytes bruts en base64 de la transaction corrompue. Rempli lorsque la transaction est (partiellement) lisible.

rawInfo (string, obligatoire)

Informations supplémentaires provenant du FDM, par exemple un message d'erreur.

}

]

}

fdmLocalisation

Objet obligatoire lié à la localisation du FDM au moment de la communication. Si le FDM contient un module 4G/5G, ces coordonnées sont obligatoires.

{

geoLocation (objet, optionnel)

Contient les données de localisation du module GPS en option du FDM

{

latitude (numérique, obligatoire)

Contient des données de latitude, au format décimal. Valeur minimale = -90, valeur maximale = 90. Format = 2,6. Exemple : 50.851415.

longitude (numérique, obligatoire)

Contient les données de longitude, au format décimal. Valeur minimale = -180, valeur maximale = 180. Taille = 3,6. Exemple : 4.354365.

}

}